

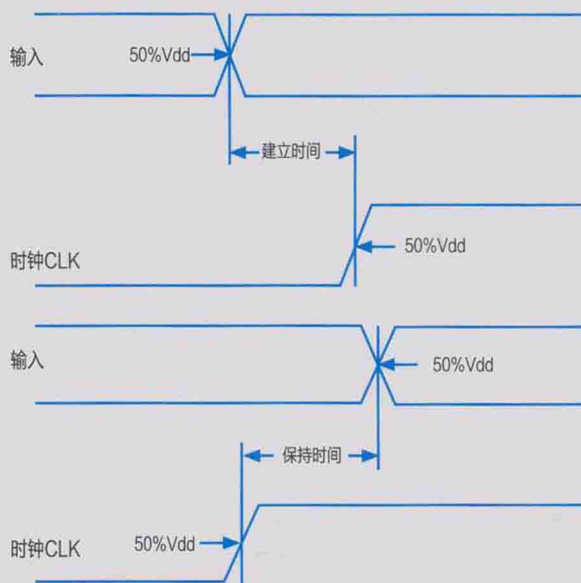
本书是系统讲解集成电路静态时序设计的经典之作。

本书作者是集成电路设计专家，拥有10年以上集成电路设计经验。

本书围绕静态时序分析和静态时序建模两方面详细介绍静态时序设计的整个流程。



电子与嵌入式系统
设计丛书



CMOS

集成电路静态时序分析与建模

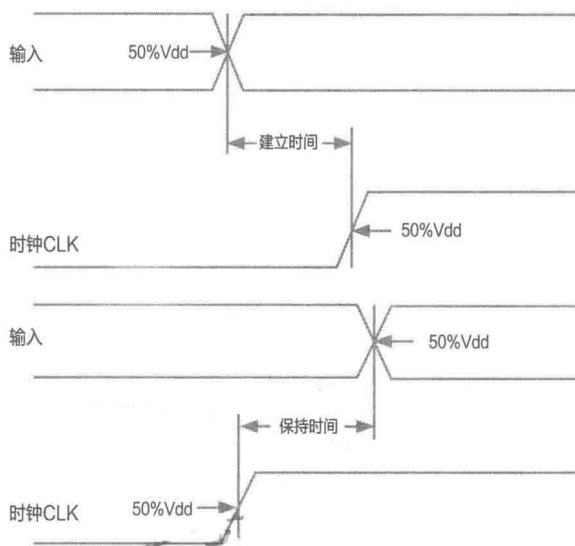
刘峰 编著



机械工业出版社
China Machine Press



电子与嵌入式系统
设计丛书



CMOS

集成电路静态时序分析与建模

刘峰 编著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

集成电路静态时序分析与建模 / 刘峰编著. —北京: 机械工业出版社, 2016.5
(电子与嵌入式系统设计丛书)

ISBN 978-7-111-53777-9

I. 集… II. 刘… III. 集成电路—研究 IV. TN4

中国版本图书馆 CIP 数据核字 (2016) 第 103726 号

集成电路静态时序分析与建模

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 缪杰 张梦玲

责任校对: 殷红

印刷: 三河市宏图印务有限公司

版次: 2016 年 6 月第 1 版第 1 次印刷

开本: 186mm × 240mm 1/16

印张: 20.25

书号: ISBN 978-7-111-53777-9

定价: 79.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88379426 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

前 言

在当今信息化的社会中，集成电路已成为各行各业实现信息化、智能化的基础。无论是军用还是民用，它都起着不可替代的作用。集成电路产业是全球范围内的核心高科技产业之一，具有战略性和市场性双重特性。在国防和国家安全领域，集成电路起着维护国家利益、捍卫国家主权安全的关键作用；在经济建设和增强综合国力的过程中，集成电路又是核心竞争力的具体表现。自 20 世纪中期以来，集成电路产业遵循摩尔定律飞速发展。集成电路产业的兴起奠定了现代信息技术的基础，现代信息技术正在迅速地改变世界及人们的生活方式，没有半导体技术及集成电路突飞猛进的发展，就没有信息技术日新月异的变化。

集成电路静态时序分析与建模技术是集成电路设计中的关键技术，高性能级芯片都需要先进的静态时序分析与建模技术来支撑。

全书共 11 章，其中静态时序分析与建模的详细内容通过第 2 ~ 11 章进行讲解。

□ 第 1 章 引论

简单介绍集成电路发展状态和重要性，使读者对集成电路行业有初步了解。同时，简单介绍业界主流的静态时序分析与建模的设计工具，使读者对静态时序有更广阔的视野。

□ 第 2 章 静态时序分析的基础知识

随着芯片尺寸的减小、集成度密集化的增强、电路设计复杂度的增加、电路性能要求的提高等，对芯片内的时序分析提出了更高的要求。静态时序分析是大规模集成电路设计中非常重要的一个方面，想熟练掌握静态时序分析，需要从掌握最基本的时序分析概念开始。因此，这一章向读者介绍静态时序分析的基本知识。

□ 第 3 章 单元库时序模型

在时序分析过程中，对一个复杂芯片中的每一个模块，不论是简单的标准单元（如 NAND、NOR 等），还是复杂的定制化设计模块（如 RAM 或处理器核），都需要有一个时序模型。因此，这一章向读者介绍单元库时序模型的基本理论。

□ 第 4 章 时序信息库文件

时序信息库文件中记录着逻辑门延时、输出信号转换延时和功耗等信息，这些信息在时

序分析时被调用,以计算电路延时值和功耗值。时序文件的内容主要由库组、属性和因子等组成。因此,这一章向读者介绍时序信息库文件中的主要基本理论。

□ 第5章 静态时序分析的基本方法

时序分析的目的在于验证设计是否符合规定时序约束下的性能要求,同时设计者基于时序分析的结果,决定如何在不满足时序要求的情况下进行时序性能改进。为了熟练地通过时序分析结果找到关键的时序违反路径并进行优化,就必须了解时序分析的基本方法。因此,这一章向读者介绍静态时序分析的基本方法。

□ 第6章 时序约束

静态时序分析工具以时序约束作为判断设计中的时序是否满足设计要求的标准,因此设计者需要提供正确的时序约束信息,以便静态时序分析工具输出正确的时序报告。通过SDC可以使用简单而又直接的方法来描述未来设计中芯片工作时所预期的时序约束,SDC的格式也是一种业界标准。因此,这一章向读者介绍主要的时序约束理论知识。

□ 第7章 串扰噪声

集成电路进入超深亚微米设计后,串扰噪声已经成为影响芯片功能和性能的重要原因之一,因此合理解决串扰噪声带来的不利影响是当前集成电路设计必须面对的越来越重要的问题。因此,这一章向读者介绍时序分析中串扰噪声的基本理论和相关的分析技术。

□ 第8章 单元时序建模实战

原理图和版图设计完成后,时序分析工具需要读取该设计的时序信息,如综合工具需要知道模块的逻辑功能、单元实际的输入负载电容、不同输入斜率和输出负载情况下单元的延时与功耗,以及单元的面积等,单元时序信息特征化就是用模拟仿真器来提取设计模块有关以上信息的过程。通过时序信息特征化来提供设计模块的时序数据,以供多种时序分析工具使用。因此,这一章向读者介绍标准单元特征化设计在实际工程应用中的基本设计技术。

□ 第9章 静态时序分析实战(ETS篇)

这一章基于OpenSparcT1里浮点计算单元部件的设计流程来讲解静态时序分析过程,向读者介绍静态时序分析在实际工程应用中的基本设计技术。

□ 第10章 Tcl脚本编程

在静态时序分析中,可以通过编程的手段来达到提高工作效率和质量的目的,时序分析工具都提供Tcl编程接口。这一章向读者介绍Tcl脚本编程的基本理论。

□ 第11章 Tcl脚本编程应用实例(PT篇)

本章将基于Synopsys公司的PrimeTime工具来讲解静态时序分析中的Tcl脚本应用实例,通过8个Tcl脚本实例由浅入深地讲解如何编写实用的Tcl脚本,从而使读者具备通过

编写 Tcl 脚本提高工作效率和编程质量的基本能力。

致谢

在此要感谢所有对本书的创作和修改做出贡献的人。

非常感谢我的导师张民选教授和李少青研究员传授给我集成电路设计知识。

非常感谢 **icdream** 论坛版主吴占托先生为本书实战素材的运行环境提供的大力支持。

非常感谢机械工业出版社对本书出版给予的大力支持。

集成电路设计领域的研究发展迅速，对于许多问题，作者并未做深入研究，一些有价值的新内容也不能及时收入本书，加上作者知识水平和实践经验有限，书中难免存在不足之处，敬请读者批评指正。

目 录

前 言

第 1 章 引论 1

- 1.1 集成电路发展史简介 1
- 1.2 国内集成电路的发展现状 2
- 1.3 国际集成电路的发展态势 4
- 1.4 静态时序分析技术 4
 - 1.4.1 静态时序分析简介 4
 - 1.4.2 静态时序分析背景 4
 - 1.4.3 静态时序分析的优缺点 5
- 1.5 主流静态时序分析与建模工具介绍 6

第 2 章 静态时序分析的基础知识 9

- 2.1 逻辑门单元 9
- 2.2 门单元的时序计算参数 10
- 2.3 时序单元相关约束 12
- 2.4 时序路径 14
- 2.5 时钟特性 17
- 2.6 时序弧 19
- 2.7 PVT 环境 24
- 2.8 时序计算单位 28

第 3 章 单元库时序模型 29

- 3.1 基本时序模型简介 29

- 3.2 Synopsys 工艺库模型 33
- 3.3 延时计算模型 38
- 3.4 互连线计算模型 45
 - 3.4.1 互连线计算模型 45
 - 3.4.2 线负载时序模型 47
- 3.5 引脚电容值的计算 49
- 3.6 功耗模型的计算 50
- 3.7 时序信息建模基本方法 51

第 4 章 时序信息库文件 54

- 4.1 非线性延时模型 54
 - 4.1.1 库组 54
 - 4.1.2 因子 57
 - 4.1.3 输入电压组 59
 - 4.1.4 输出电压组 59
 - 4.1.5 功耗查找表模板组 59
 - 4.1.6 操作条件组 60
 - 4.1.7 线负载组 60
 - 4.1.8 延时查找表模板组 61
 - 4.1.9 单元组 62
 - 4.1.10 引脚组 64
 - 4.1.11 触发器组 67
 - 4.1.12 逻辑状态表组 68
 - 4.1.13 电源引脚组 69
 - 4.1.14 延时组 69

| | | | |
|--------------------------------|-----|--|-----|
| 4.1.15 单元上拉延时组 | 70 | 6.1.3 虚拟时钟 | 114 |
| 4.1.16 单元下拉延时组 | 71 | 6.1.4 最小时钟脉宽 | 116 |
| 4.1.17 上拉转换组 | 71 | 6.2 I/O 延时约束 | 117 |
| 4.1.18 下拉转换组 | 72 | 6.3 I/O 环境建模约束 | 119 |
| 4.1.19 上拉约束组 | 72 | 6.4 时序例外 | 121 |
| 4.1.20 下拉约束组 | 73 | 6.5 恒定状态约束 | 125 |
| 4.1.21 内部功耗组 | 73 | 6.6 屏蔽时序弧 | 126 |
| 4.1.22 哑阈漏流功耗组 | 74 | 6.7 时序设计规则约束 | 127 |
| 4.2 复合电流源延时模型 | 75 | 第 7 章 串扰噪声 | 129 |
| 4.2.1 输出电流查找表模板组 | 75 | 7.1 噪声的定义 | 129 |
| 4.2.2 输出上拉电流组 | 75 | 7.2 噪声的来源 | 130 |
| 4.2.3 输出下拉电流组 | 76 | 7.3 噪声恶化的原因 | 133 |
| 4.2.4 向量组 | 76 | 7.4 噪声的体现形式 | 134 |
| 4.2.5 接收电容组 | 77 | 7.5 噪声相互作用形式 | 135 |
| 第 5 章 静态时序分析的基本方法 | 79 | 7.6 NLDM 噪声模型的计算 | 136 |
| 5.1 时序图 | 79 | 7.7 噪声延时计算方法 | 141 |
| 5.2 时序分析策略 | 80 | 7.8 时间窗口 | 143 |
| 5.3 时序路径延时计算方法 | 81 | 7.9 优化噪声的物理方法 | 145 |
| 5.4 时序路径的分析方法 | 83 | 7.10 CCS 噪声模型 | 148 |
| 5.5 时序路径分析模式 | 88 | 第 8 章 单元时序建模实战 | 153 |
| 5.5.1 单一分析模式 | 90 | 8.1 时序信息提取实现 | 153 |
| 5.5.2 最好 - 最坏分析模式 | 91 | 8.1.1 时序信息特征化实现流程 | 153 |
| 5.5.3 芯片变化相关分析模式 | 94 | 8.1.2 时序信息特征化数据准备 | 154 |
| 5.6 时序减免 | 96 | 8.1.3 标准单元时序信息提取 | 158 |
| 5.7 其他芯片变化相关分析模式 | 98 | 8.2 SiliconSmart 工具的使用流程简介 | 162 |
| 5.8 时钟路径悲观移除 | 103 | 8.3 时序信息提取内容 | 163 |
| 5.9 时序优化 | 105 | 第 9 章 静态时序分析实战 (ETS 篇) | 170 |
| 第 6 章 时序约束 | 107 | 9.1 静态时序分析的基本流程 | 170 |
| 6.1 时钟约束 | 107 | 9.2 建立静态时序分析的工作环境 | 171 |
| 6.1.1 创建时钟 | 107 | | |
| 6.1.2 生成时钟 | 111 | | |

| | | | |
|------------------------------|-----|---|-----|
| 9.3 静态时序分析实现 | 174 | 10.7 过程 | 223 |
| 9.3.1 建立时间分析 | 174 | 10.7.1 过程定义和返回值 | 224 |
| 9.3.2 保持时间分析 | 192 | 10.7.2 局部变量和全局变量 | 224 |
| 9.3.3 时序设计规则分析 | 201 | 10.7.3 默认参数和可变个数 参数 | 225 |
| 9.3.4 时序违反修复 | 204 | 10.8 引用 | 226 |
| 第 10 章 Tcl 脚本编程 | 207 | 10.9 字符串操作 | 228 |
| 10.1 Tcl 语法 | 207 | 10.10 文件访问 | 234 |
| 10.1.1 命令格式 | 207 | 10.10.1 文件名 | 234 |
| 10.1.2 替换 | 209 | 10.10.2 基本文件输入 / 输出 命令 | 234 |
| 10.1.3 双引号和花括号 | 211 | 第 11 章 Tcl 脚本编程应用实例 (PT 篇) | 237 |
| 10.1.4 注释 | 211 | 11.1 get_failing_paths_high_slew | 237 |
| 10.2 数据结构 | 212 | 11.2 get_interclock_skew | 241 |
| 10.2.1 简单变量 | 212 | 11.3 report_unclocked | 244 |
| 10.2.2 数组 | 212 | 11.4 get_buffers | 248 |
| 10.3 表达式 | 212 | 11.5 get_ports_edge_sense | 255 |
| 10.3.1 操作数 | 213 | 11.6 report_clock_endpoint_skew | 260 |
| 10.3.2 运算符和优先级 | 213 | 11.7 report_violations | 264 |
| 10.3.3 数学函数 | 214 | 11.8 eco_fix_violations | 271 |
| 10.3.4 列表集合 | 215 | 附录 | 290 |
| 10.4 控制流 | 219 | 参考文献 | 313 |
| 10.4.1 if 命令 | 219 | | |
| 10.4.2 循环命令 | 220 | | |
| 10.5 eval 命令 | 223 | | |
| 10.6 source 命令 | 223 | | |

第 1 章

引 论

集成电路 (Integrated Circuit, IC) 对一般人来说也许会有陌生感, 但其实我们和它打交道的机会很多, 如计算机、电视机、手机、网站、取款机等中都含有集成电路, 数不胜数。除此之外, 在航空航天、星际飞行、医疗卫生、交通运输、武器装备等许多领域中, 几乎都离不开集成电路的应用。在当今世界, 说它无孔不入并不过分。

所谓集成电路, 就是在一块极小的硅单晶片上, 利用半导体工艺制作出许多二极管、晶体管及电阻、电容等, 并连接成能完成特定电子技术功能的电子电路。从外观上看, 它已成为一个不可分割的完整器件。集成电路在体积、重量、耗电、寿命、可靠性及电性能方面远远优于由晶体管组成的电路。

在当今这信息化的社会中, 集成电路已成为各行各业实现信息化、智能化的基础。无论是在军事还是在民用上, 它都起着不可替代的作用。集成电路产业是全球范围内的核心高科技产业之一, 具有战略性和市场性双重特性。在国防和国家安全领域, 集成电路起着维护国家利益、捍卫国家主权的关键作用; 在经济建设和增强综合国力的过程中, 集成电路又是核心竞争力的具体表现。自 20 世纪中期以来, 集成电路产业遵循摩尔定律飞速发展。集成电路产业的兴起为现代信息技术奠定了基础, 现代信息技术正在迅速地改变着世界及人们的生活方式, 没有半导体技术突飞猛进的发展, 就没有信息技术日新月异的变化。

1.1 集成电路发展史简介

集成电路的发展经历了一个漫长的过程, 这里以时间顺序来简单地介绍它的发展过程。

1906 年, 第一个电子管诞生; 1912 年前后, 电子管的制作日趋成熟, 引发了无线电技术的发展; 1918 年前后, 逐步发现了半导体材料; 1920 年, 发现半导体材料所具有的光敏特性; 1932 年前后, 运用量子学说建立了能带理论以研究半导体现象; 1947 年, 美国贝尔实验室的巴丁等人发明了晶体管, 如图 1-1 所示。作为划时代的发明, 他们因此获得了 1956 年诺贝尔物理学奖。

1952 年, 英国科学家 G.W.A.Dummer 第一次提出了集成电路的设想; 1958 年, TI 公司的科学家 Clair Kilby 与仙童公司的 Robert Noyce 先后独立地发明了集成电路, 如图 1-2 所

示。Kilby 等人获得了 2000 年诺贝尔物理学奖，以表彰他们为现代信息技术所做出的基础性贡献。

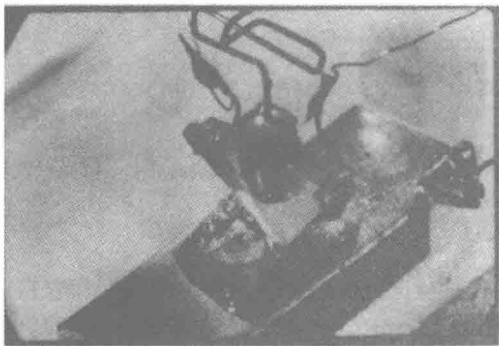


图 1-1 巴丁等人发明的晶体管

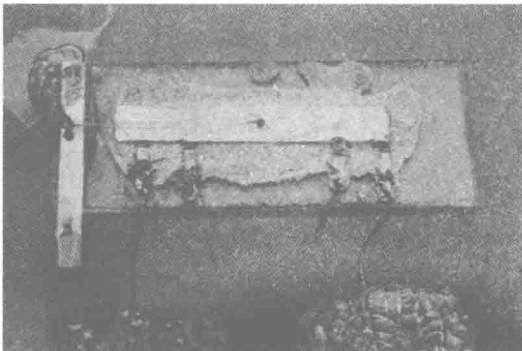


图 1-2 第一块集成电路

1966 年，美国的贝尔实验室使用比较完善的硅外延平面工艺制造成第一块公认的大规模集成电路。1971 年，Intel 公司生产出第一个微处理器芯片 4004，如图 1-3 所示。

1988 年，16M DRAM 问世，在 1cm^2 大小的硅片上集成了 3500 万个晶体管，这标志着进入了超大规模集成电路的更高阶段。1997 年，300MHz 奔腾 II 问世，它采用 $0.25\mu\text{m}$ 工艺，奔腾系列芯片的推出让计算机的发展如虎添翼，发展速度让人惊叹。2009 年，Intel 酷睿 i 系列全新推出，采用 32nm 工艺。



图 1-3 第一个微处理器芯片 4004

集成电路的集成度从小规模到大规模、再到超大规模的迅速发展，关键就在于集成电路的布图设计水平的迅速提高，集成电路的布图设计由此而日益复杂而精密。这些技术的发展，使得集成电路的发展进入了一个新的发展阶段。相信随着科技的发展，集成电路还会有更高水平的发展。

1.2 国内集成电路的发展现状

集成电路对国民经济发展的倍增作用已被人们充分认识，它可表示为：IC 业：电子整机业：应用服务业 = 1：10：100。有关资料表明，集成电路对一个国家经济发展的贡献率远大于其他产业：钢铁对国民生产总值（GNP）的贡献率为 1，汽车对 GNP 的贡献率为 5，彩电对 GNP 的贡献率为 30，计算机对 GNP 的贡献率为 1000，而集成电路对 GNP 的贡献率为 3000。发达国家 GNP 增长部分的 65% 与 IC 业相关，尤其是集成电路高级产品：芯片，它被

人们形象地比喻为国家的“工业粮食”，是信息产业的核心，是所有整机设备的“心脏”。据国际货币基金组织测算，芯片1元的产值可带动相关电子信息产业10元产值，带来100元的GDP。因此，欧美发达国家纷纷将芯片产业列入国家战略产业。

我国集成电路产业诞生于20世纪60年代，经过50多年的发展，目前已形成一定的发展规模，由7个芯片生产骨干企业，十几个重点封装厂，几十家设计公司，若干个关键材料及专用设备仪器制造厂组成的产业群体初步形成，电路设计、芯片制造和电路封装三业并举。

国内集成电路产业最近几年获得了较快的发展，一些优势企业的竞争力开始显现。以增长最快的设计业为例，2011年，IC设计业销售额达到473.74亿元，同比增长30.2%；2012年，我国设计企业前10家的销售额总和达到231.17亿元，比上年增加29.7亿元，10家企业的销售额总和占全行业销售额总和的33.97%，比上年（31.76%）增加2.21%。但是，打价格战还是主要的商业策略，“正向设计”依然未形成主流，基础能力提升不快的状况仍未改观。全行业的销售额总和有可能还低于世界排名第一的设计企业销售额。近十几年来，我国大陆境内IC制造生产线快速增加，“十·五”期间增加了16条，“十一·五”期间增加了20条，大尺寸线在总量中所占比例也在逐步上升。2011年，虽然芯片制造业同比仍有8.9%的增速，规模达到486.91亿元，但销售收入增速出现明显回落。目前，国内芯片制造厂有近50家，具有200mm及300mm的纯芯片代工企业已有7家：中芯国际、华虹NEC、上海宏力、海力士一意法半导体（无锡）、和舰科技、台积电（上海）、上海先进，形成了颇具规模的企业群体和生产能力。

需要特别指出的是，虽然我国大陆境内IC制造生产线快速增加，但其投资主体多为外资转移的生产线或中外合资合作，并非国有控股，涉及国家安全的战略性支撑缺失。国防军用IC开发自主可控绝对是必要的。对IC的“信任”不可能在设计或制造之后加入。军品所需的安全性、及时性、高可靠、抗极端环境等要求是“委外代工”或“民用代工”无法达到的。在设计、制造和封装测试三大生产环节中，封测可能是技术含量相对较低的部分，也是最靠近市场的后端生产环节。因此，国（境）外封装测试代工企业大量地将其封装产能转移至中国大陆境内，直接促进了我国封测产业规模的迅速扩大。2010年，国内集成电路产业销售收入规模为1440.2亿元，其中，封测业销售收入规模为639亿元，占集成电路产业总销售收入的比重约为44%。

根据邓中翰院士的观点，近几年尽管我国取得了可喜的进步，但是我国每年进口芯片仍然高达2000多亿美元。据海关总署在2014年1月公布的数据，2013年全年我国集成电路进口额为2322亿美元，比上年同期的1724.99亿美元增长了34.6%；逆差达到1441亿美元，较上年同期的1391亿美元扩大了50亿美元，连续第4年扩大。而同2013年相比，2014年虽然中国原油进口量继续大幅增长，但进口额度只有2196亿美元，较前一年同期下降了0.5%。这也说明，2014年我国芯片进口额度与石油进口额度的比例也在进一步扩大。

1.3 国际集成电路的发展态势

根据国际半导体技术发展路线图 (ITRS)，国际集成电路技术大致有 3 个主要趋势：

1) 延续摩尔定律，继续按比例缩小，Intel CMOS 技术已达到 22nm 工艺节点，于 2013 年引入 14nm 工艺节点，且正在部署 7nm。台积电最高端的 CMOS 已达到 28nm。

2) 功能集成（称为拓展摩尔定律），即在单个芯片 / 封装 / 模块上更多地集成 RF、功率控制、无源器件等功能单元。

3) 发展新兴材料和器件，预计到 2019 年，可研究出超过 CMOS 器件性能的新器件，继续提高 CMOS 工艺的能力。

1.4 静态时序分析技术

随着芯片尺寸的减小和集成度密集化的增强、电路设计复杂度的增加、电路性能要求的提高等，新的变化都对芯片内的时序分析提出了更高的要求。静态时序分析是大规模集成电路设计中非常重要的一个环节，它能验证设计在时序上的正确性，并决定设计是否能够在要求的工作频率下运行。

1.4.1 静态时序分析简介

静态时序分析 (static timing analysis, STA) 是分析、调试并确认一个门级系统设计时序性能的比较彻底的方法。在门级电路设计过程中，为得到一个最佳的电路设计，在结构逻辑、电路布局布线等方面，时序分析起着关键性的作用。静态时序分析既要检验门级电路的最大延迟、以保证电路在指定的频率下能够满足建立时间的要求，同时又要检验门级电路的最小延迟、以满足保持时间的需求。芯片的设计只有通过了静态时序分析才能真正完成，甚至在从逻辑综合开始后的每一个设计步骤的结果都需要满足或部分满足时序的要求。

1.4.2 静态时序分析背景

集成电路设计初期流程中，设计者使用动态仿真来验证整个设计或部分设计的功能和时序。动态仿真需要专门设计的仿真向量来检验设计中的时序关键路径和时序信息。这种方法根据芯片的动态时序行为使用输入向量来检验功能路径。基于动态仿真的方法既能够验证设计的功能，也能够验证设计的时序，这是一种非常流行的时序验证策略。随着设计规模的增大，验证一个设计需要测试的向量的数量按指数级增长，而且这种方法难以保证足够的覆盖率。在大型设计中，如果仅用传统的动态仿真方法，则所需时间及工作量都难以承受。目前，设计人员在整个设计周期中需要花费 50% 以上的时间来执行设计的功能和时序验证。设计人员必须为验证创建独立时序向量和功能向量。创建能够彻底地检验设计中每一条路径的时序向量是非常困难的。随着设计尺寸和复杂性的增加，并且由于上市时间的压力导致整个

设计周期缩短, 向量产生的问题逐渐显现出来。现存的仿真工具并没有足够的性能和能力对几百万门的设计进行完整的时序仿真。更大设计的出现以及庞大的向量集合使得动态仿真在设计流程中成为一个严重的瓶颈。上市时间的压力、芯片的复杂度、传统仿真器的仿真速度和计算能力上的限制都促进了时序分析技术从动态向静态迁移。

1.4.3 静态时序分析的优缺点

时序分析是评定一个集成电路设计在特定工作频率上正常工作的能力, 尽管这样的时序测量在理论上能够使用动态仿真来实现, 但动态仿真的方法在实际使用中非常耗时。静态分析由于具备如下优点而在测量和优化电路时序方面扮演着重要的角色。

静态时序分析的优点如下。

1) 静态时序分析执行速度快。

高性能集成电路通常根据其工作的时钟频率来描述其特性, 设计者常常将工作时钟频率作为高性能的集成电路的特性之一。为了测试电路在指定速率下运行的能力, 设计者需要在设计过程中测量电路在不同工作阶段的延迟。此外, 在不同的设计阶段(如逻辑综合、布局、布线等)需要对时间优化程序内部进行延迟计算。尽管可以通过严格的 SPICE 电路仿真来进行此类时间测量, 但是这种方法在实用中会耗费大量时间。静态时序分析在电路时序快速、准确的测量中扮演了重要角色。

2) 静态时序分析不需要测试向量。

静态时序分析之所以能够更快速地完成任务, 是因为它使用了简化的模型, 而且它有限地考虑了信号之间的逻辑互动, 同时不依赖于激励且能穷尽分析所有的时序路径。它不仅比动态仿真运行速度快, 占用内存少, 而且能分别在版图前、后根据仿真模型以及提取的寄生参数对各种时序路径进行检测。由于静态时序分析的方法不需要任何测试向量, 所以所需要的时间远远少于门级动态仿真。

3) 静态时序分析对于有时序路径的时序, 测试覆盖率可以近乎达到 100%。

静态时序分析可以降低验证的复杂性, 同时静态时序分析提供一种针对大规模设计验证的有效解决方法, 它将整个设计分解成一系列路径的集合, 然后以此为基本单位, 分析并报告每条路径的延时是否满足时序约束要求。而在动态仿真中, 为了达到相同的分析覆盖率所需的仿真向量是极其庞大的, 也是不可能完成的。由于静态时序分析并不基于测试向量, 所以典型情况下非常快速, 并且能够适应非常大的设计, 同时很好地实现了近乎 100% 的约束覆盖率。

4) 静态时序分析能够完成动态仿真所不能实现的复杂分析。

静态时序分析可以识别的时序故障数量要远多于动态仿真, 例如最大和最小延时路径分析, 建立和保持时间分析, 时钟信号质量分析, 报告不受时序约束的逻辑路径等。

静态时序分析的优点并不意味着它能够完全替代动态仿真。静态时序与动态仿真相比, 也有自身的缺点, 静态时序验证工具与动态仿真验证工具必须协同存在。

静态时序分析的缺点如下：

- 1) 静态时序分析不能验证设计的功能，设计功能验证还必须使用功能仿真来实现。
- 2) 静态时序分析只能验证同步时序电路的时序特性，如果设计中含有较多的异步电路，则应该通过门级动态仿真来验证。
- 3) 静态时序分析不能自动识别设计中的特殊路径，比如多周期路径（multi-cycle path）、非正常路径（false path）、多时钟分配关系（multiple clock）等，这需要设计者手动地通过时序约束文件来指导静态时序分析。

1.5 主流静态时序分析与建模工具介绍

集成电路设计中业界主要使用 Cadence 公司和 Synopsys 公司的时序分析工具来完成大部分的时序分析工作。

1. Encounter Timing System

Cadence 公司的 Encounter Timing System 为客户提供了面向时序、信号完整性（SI）和功耗的统一数据源——从设计和物理实现，到最后的签收（signoff）分析。它不仅满足了实现和签收分析的需要，前端设计团队还可以利用其全局时序调试功能，实现精确的根源分析和迅速的时序收敛，并且它还拥有着强大的图形用户界面。通过 Encounter Timing System，数字 IC 设计师可以克服不断缩小的工艺节点带来的困难，缩短上市时间、提高效率，将信号完整性分析应用到设计流程的各个方面，并降低总生产成本。Cadence 公司的 Encounter Timing System 具备了 CeltIC NDC 领先的信号完整性分析和悲观剔除技术（pessimism removal）的全部优点，具备了达到签收质量的时序、延迟计算、电源完整性等功能，并且与 Encounter Conformal 技术紧密结合，以在设计流程所有阶段得到全局、系统级的时序视图。其他功能包括关键路径模拟、SPICE 追踪、电迁移分析、统计时序，以及计算功耗优化及低功耗设计架构的能力。

Cadence 公司的 Encounter Timing System 工具如图 1-4 所示。

2. PrimeTime

Synopsys 公司的 PrimeTime 是针对复杂、亿万门芯片进行全芯片、门级静态时序分析的工具。PrimeTime 可以集成逻辑综合和物理综合的流程，让设计者分析并解决复杂的时序问题，并提高时序收敛的速度。PrimeTime 是众多半导体厂商认可的、业界标准的静态时序分析工具。PrimeTime 强大的性能得益于对生成报告和基于标准延迟文件（SDF）的时序分析方面的算法的改进。PrimeTime 提供全芯片级的静态时序分析，同时整合了延迟计算和先进的建模功能，以实现有效而又精确的时序认可。PrimeTime SI 是全芯片门级信号完整性分析工具。PrimeTime SI 建立在成功流片验证过的 PrimeTime 平台之上，提供精确的串扰延迟分析、IR Drop（电压降落）分析和静态时序分析。PrimeTime 业界领先的超快运行时间和处理容量让数亿万门的复杂设计一次流片成功，缩短芯片进入市场的时间。

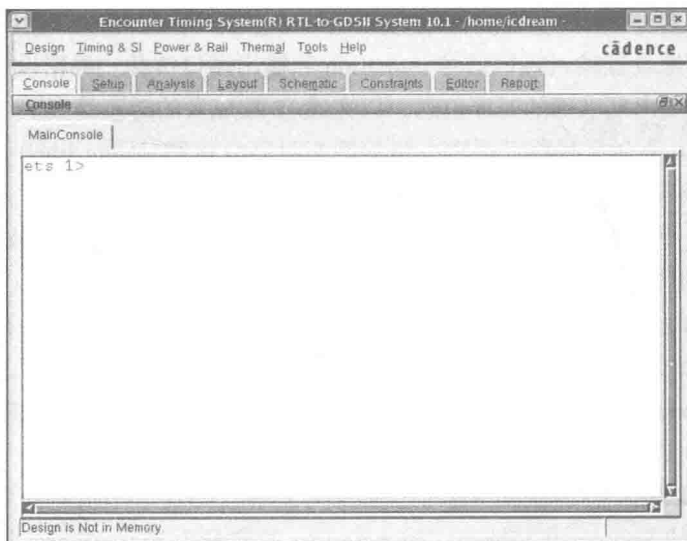


图 1-4 Encounter Timing System 工具界面

Synopsys 公司的 PrimeTime 工具如图 1-5 所示。

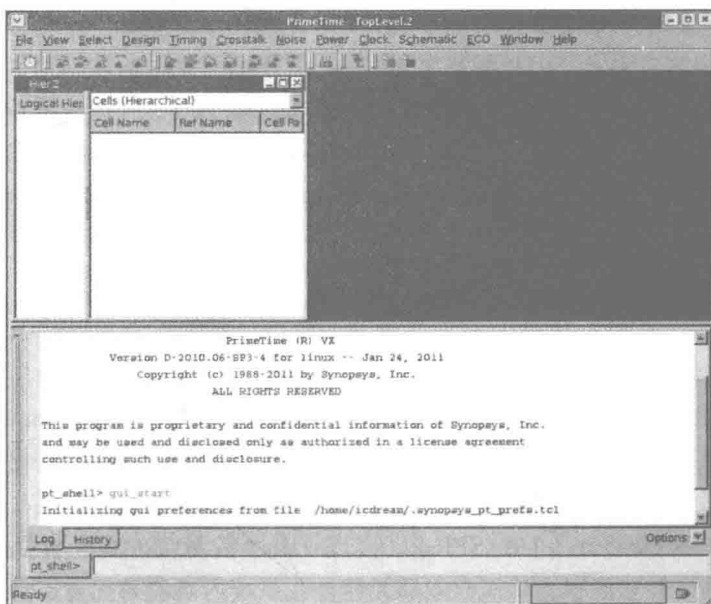


图 1-5 PrimeTime 工具界面

3. Nanotime

NanoTime 是 Synopsys 公司新一代的晶体管级静态时序分析工具。它集对晶体管级全定制模块电路完整的静态时序验证（包括 SI 分析）和产生该模块电路 .lib 文件等强大功能于一

身。通过自动生成合乎 Liberty 语法规则格式的 timing library 文件, IP 设计用户可以根据应用实际情况,自由地设置输入端口信号 transition 和输出端口负载。根据设置值在 .lib 文件中查表即可获取该模块内部准确的时序信息,以完成 IP 集成后的时序检查。

NanoTime 定位于全定制数字逻辑电路的静态时序分析和 timing library 生成,因此其拥有强大的逻辑电路拓扑识别能力,可自动识别诸如 inverter、mux、xor、nand、nor、clock-gate、turnoff、cross_coupled、latch、precharge、feedback、weak_pullup 等大多数常见的电路结构,对于部分复杂的电路,如 latch、ram、flip-flop,可以借助某些命令加以识别,如 mark_latch、mark_flip_flop、mark_register_file 等。在识别这些逻辑电路并设置了时序约束和工作条件后,NanoTime 会提取所有的数据和时钟路径的延迟信息(trace path),并会根据客户设置的时序约束(constrain)对内部的时序电路,如 latch、DFF 等进行 setup/hold、recovery/remove 等检查。

4. Encounter Library Characterizer

Cadence 公司的 Encounter Library Characterizer 的前身为 SignalStorm Library Characterizer,简称 SLC,它能够自动生成最新建模格式的单元时序库。Library Characterizer 可以使用高效电流源模型(ECSM)来对时序、功耗、信号完整性等进行建模,同时也可支持 CCS 模型精确地进行噪声、多电压、电压降的建模,并用于时序和功耗分析。

5. Liberty NCX

Synopsys 公司的 Liberty NCX 针对不同的电压、温度等环境参数或自创的 Cells 来进行时序特征化提取。Liberty NCX 是一种取代 Synopsys 前特征化工具 NanoChar 的新型库特征化解决方案,它包括一个模型特征化引擎、一套库质量校验器及模型转换等功能。此外,Liberty NCX 还包含同时特征化和模型精确度验证能力。Liberty NCX 面向代工厂、IDM 或 IP 提供商的内部库开发组织。

6. Siliconsmart

Synopsys 公司的 Siliconsmart 具有业界领先的精度、吞吐量和易用性,它提供了适用于所有主流设计流程的标准单元、I/O、定制宏和存储器特征化与建模,支持所有先进时序、功耗、噪声和统计建模规格。其专有的加速电路引擎(accelerated circuit engine,ACE)全面自动化了特征化流程,部署了最先进的电路功能识别技术及矢量生成和优化算法,可在无损精度的前提下有效完成单元特征化。为获得最高性能,Siliconsmart 还嵌入了精确且超快速的 SPICE 仿真器——FineSim(TM) SPICE,实现了几个数量级的吞吐量提升。此外,它还包括了一个闭环模型验证流程,让用户能够无缝地在系统内启动第三方工具来验证所生成的模型。

第 2 章

静态时序分析的基础知识

随着芯片尺寸的减小和集成度密集化的增强、电路设计复杂度的增加、电路性能要求的提高等，每一个新的变化都对芯片内的时序分析提出了更高的要求。静态时序分析是大规模集成电路设计中非常重要的一个方面，熟练掌握静态时序分析需要从掌握最基本的时序分析概念开始。

2.1 逻辑门单元

逻辑门单元是实现基本逻辑运算和复合逻辑运算的单元电路。逻辑门单元分为组合逻辑门单元和时序逻辑门单元两种。CMOS 工艺下，数字电路逻辑门单元主要由 PMOS 和 NMOS 晶体管构成。反相器逻辑门单元如图 2-1 所示。

反相器逻辑门的逻辑功能是根据输入数据的逻辑电平进行逻辑取反的求值运算，并通过输出导出求值结果。其输出结果有两个状态：高电平和低电平，分别对应数字逻辑的 1 和 0。

时序分析中逻辑门延时和信号线延时这两个参数值组成的阶段延时（stage delay）是时序分析中计算延时的主要组成部分，如图 2-2 所示。

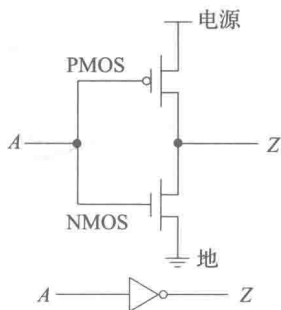


图 2-1 反相器逻辑门单元

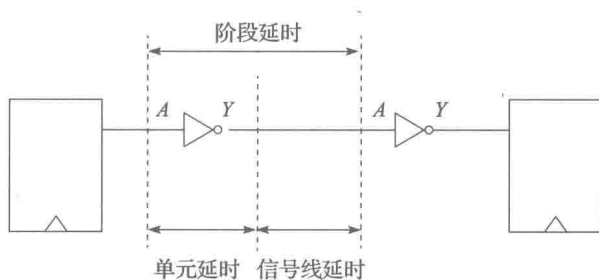


图 2-2 阶段延时组成

逻辑门延时定义为逻辑单元自身逻辑求值的时间。不同的逻辑门单元对应不同的逻辑门延时，那么反相器逻辑门单元自身的求值时间就对应该反相器的逻辑门延时。信号线延时的定义为：逻辑信号从逻辑门单元的输出端口开始在互连线上传播到下一级逻辑输入端口的延时。

2.2 门单元的时序计算参数

静态时序分析中,需要通过提取逻辑门单元相关的时序参数的数值来验证设计在时序上的正确性。组合逻辑门单元相关的时序参数主要包括信号转换延时和逻辑门延时。

1. 信号转换延时 (transition delay)

输入端口或者输出端口的信号电平由高电平转换成低电平,或者由低电平转换成高电平所需要的时间即为信号转换延时。

信号由低电平到高电平的信号转换延时如图 2-3 所示。

如图 2-3 所示,信号的有效高低电平值通过定义逻辑 1 阈值电压 V_{th2} 和逻辑 0 阈值电压 V_{th1} 来界定。当信号电压值大于逻辑 1 阈值电压 V_{th2} 时,其信号为高电平有效。同样,当信号电压值小于逻辑 0 阈值电压 V_{th1} 时,其信号为低电平有效。

静态时序分析中,时序信息文件中通过以下 4 个设置来定义信号转换延时的计算参数属性。

(1) slew_lower_threshold_pct_fall

该设置定义信号电压从高电平到低电平转换延时的低电平有效阈值计算电压为标准供电电压的百分比参数。

(2) slew_upper_threshold_pct_fall

该设置定义信号电压从高电平到低电平转换延时的高电平有效阈值计算电压为标准供电电压的百分比参数。

(3) slew_lower_threshold_pct_rise

该设置定义信号电压从低电平到高电平转换延时的低电平有效阈值计算电压为标准供电电压的百分比参数。

(4) slew_upper_threshold_pct_rise

该设置定义信号电压从低电平到高电平转换延时的高电平有效阈值计算电压为标准供电电压的百分比参数。

示例设置如下:

```
slew_lower_threshold_pct_fall : 20.0;
slew_upper_threshold_pct_fall : 80.0;
slew_lower_threshold_pct_rise : 10.0;
slew_upper_threshold_pct_rise : 90.0;
```

以上设置定义了逻辑门的信号电压从高电平到低电平转换延时为信号电压从标准供电电压 80% 转换到标准供电电压 20% 的时间间隔、逻辑门的信号电压从低电平到高电平转换延时为信号电压从标准供电电压 10% 转换到标准供电电压 90% 的时间间隔。逻辑门信号的有效电压定义如图 2-4 所示。

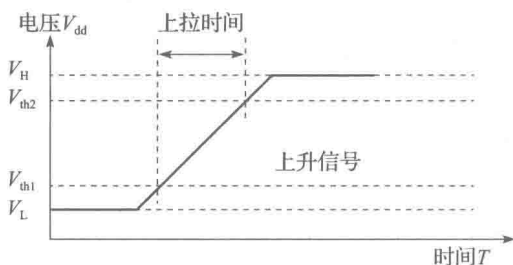


图 2-3 信号转换延时

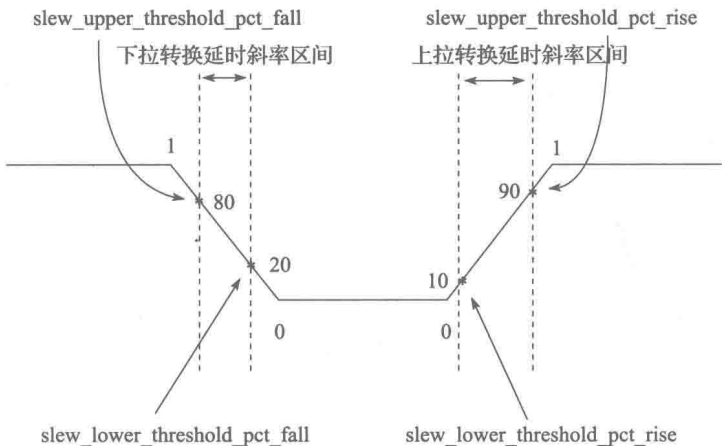


图 2-4 逻辑门信号的有效电压定义

根据设计的不同需求，设计者可以通过这 4 个参数来设置不同的信号有效阈值电压百分比参数。

2. 逻辑门延时 (logic gate delay)

通过由晶体管组成的逻辑门可以使高、低电平输入信号进行逻辑求值并产生对应的高电平或者低电平信号输出，而从输入信号改变到对应的正确输出信号的时间间隔即为逻辑门延时。反相器门延时如图 2-5 所示。

静态时序分析中，在时序信息文件内通过以下 4 个设置来定义逻辑门延时的计算参数属性。

(1) input_threshold_pct_rise

该设置定义高电平输入信号的高电平计算阈值电压为标准供电电压的百分比参数。

(2) output_threshold_pct_rise

该设置定义高电平输出信号的高电平计算阈值电压为标准供电电压的百分比参数。

(3) output_threshold_pct_fall

该设置定义低电平输出信号的的低电平计算阈值电压为标准供电电压的百分比参数。

(4) input_threshold_pct_fall

该设置定义低电平输入信号的的低电平计算阈值电压为标准供电电压的百分比参数。

示例设置如下：

```
input_threshold_pct_fall : 60;
output_threshold_pct_fall : 50;
```

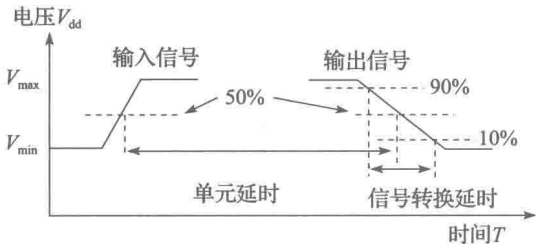


图 2-5 反相器门延时

以上设置定义逻辑门从低电平输入信号到低电平输出信号的延时为：从输入信号低于标

准供电电压 60% 开始到输出信号变化到低于标准供电电压 50% 时的时间间隔。逻辑门信号延时定义如图 2-6 所示。

同样，通过这 4 个设置还可以定义逻辑门在不同信号输入到对应输出的延时参数，如从高电平输入信号到低电平输出信号的延时参数、从高电平输入信号到高电平输出信号的延时参数，以及从低电平输入信号到高电平输出信号的延时参数。

2.3 时序单元相关约束

相比组合逻辑单元，时序单元除了具有组合逻辑单元的时序参数属性，还存在更多的时序约束参数属性，这些都是时序单元特有的相关时序约束。时序单元的时序约束是为了保证时序单元能够实现正确的逻辑功能所规定的输入或者输出信号数据需要保持稳定的最小时间间隔值。时序单元相关约束信息包括：建立时间、保持时间、恢复时间、移除时间及最小脉冲宽度。

1. 建立时间

对于时序单元，建立时间是指时序单元正常工作时，在功能上为了保证正确性，输入信号数据应该在时钟信号有效前到达并保持的最小时间。根据前面介绍的单元时序计算参数设置，计算建立时间的有效值需要根据单元时序计算参数的设置进行。根据设计的需求，设计者可以设置不同的单元时序计算百分比参数，一般设置为 50%。

如图 2-7 所示，建立时间的测量值定义为：从数据信号的电压达到标准供电电压 50% 时间点到时钟信号电压达到标准供电电压 50% 时间点的時間间隔。

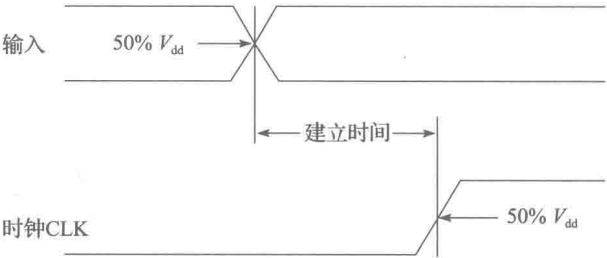


图 2-7 建立时间的定义

2. 保持时间

对于时序单元，保持时间是指时序单元要实现正确的逻辑功能，数据信号在时钟沿有效

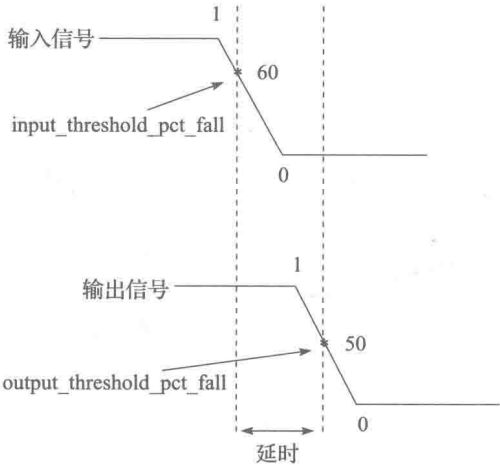


图 2-6 逻辑门信号延时定义

后必须保持的最小时间长度。同样，计算保持时间的测量值也需要根据单元时序计算参数的设置进行。根据设计的不同需求，设计者可以设置不同的单元时序计算百分比参数，一般设置为 50%。

如图 2-8 所示，保持时间的测量值定义为：从时钟信号电压达到标准供电电压 50% 时间点到数据信号的电压达到标准供电电压 50% 时间点的時間间隔。

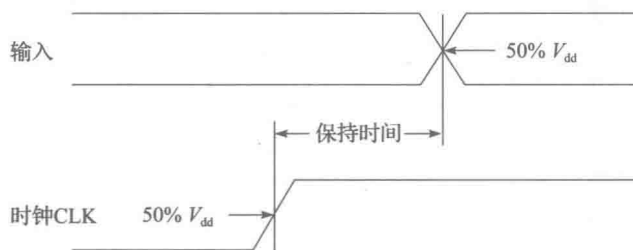


图 2-8 保持时间的定义

3. 恢复时间

恢复时间是指为了保证单元正确的逻辑功能，要求低电平复位信号或者高电平清零信号在时钟有效沿之前保持有效的最小时间长度。同样，计算恢复时间的测量值也需要根据单元时序计算参数的设置进行。根据设计的不同需求，设计者可以设置不同的时序计算百分比参数，一般设置为 50%。

如图 2-9 所示，恢复时间的测量值定义为：从复位或清零信号的电压达到标准供电电压 50% 时间点到时钟信号电压达到标准供电电压 50% 时间点的時間间隔。

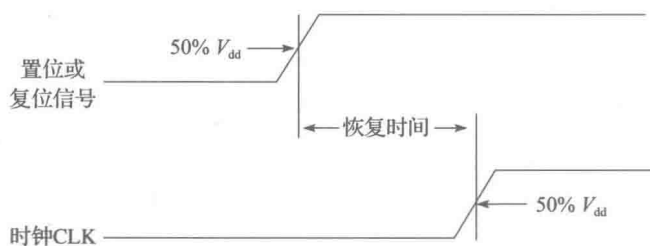


图 2-9 恢复时间的定义

4. 移除时间

移除时间是指为了保证单元正确的逻辑功能，要求低电平复位信号或者高电平清零信号在时钟有效沿之后继续保持有效的最小时间长度。同样，计算移除时间的测量值也需要根据单元时序参数设置进行。根据设计的不同需求，设计者可以设置不同的时序计算百分比参数，一般设置为 50%。

如图 2-10 所示，移除时间的测量值定义为：从时钟信号电压达到标准供电电压 50% 时间点到复位或清零信号的电压达到标准供电电压 50% 时间点的時間间隔。

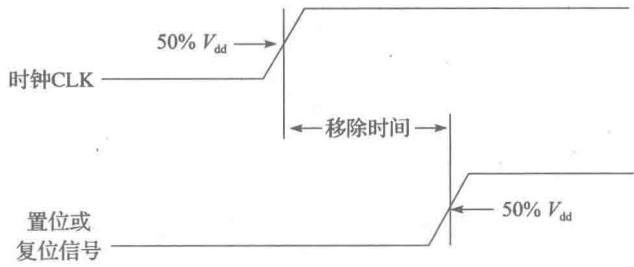


图 2-10 移除时间的定义

5. 最小脉冲宽度

最小脉冲宽度是指脉冲波形的开始到结束之间的最小时间间隔。同样，计算最小脉冲宽度的测量值也需要根据单元时序计算参数的设置进行。根据设计的不同需求，设计者可以设置不同的单元时序计算百分比参数，一般设置为 50%。

如图 2-11 所示，最小高电平脉冲宽度测量值为：信号的上升沿电压达到标准供电电压 50% 的时间点到相邻的信号电压下降沿达到标准供电电压 50% 的时间点的时间间隔。最小低电平脉冲宽度测量值为：信号的下降沿电压达到标准供电电压 50% 的时间点到相邻的信号电压上升沿达到标准供电电压 50% 的时间点的时间间隔。

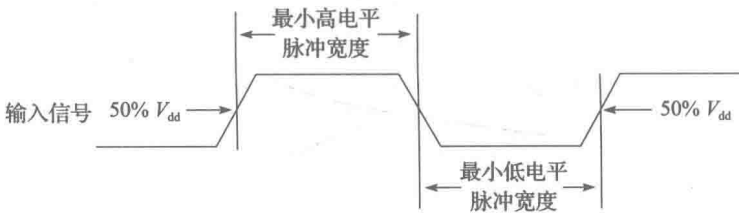


图 2-11 最小脉冲宽度的定义

时序单元的时序分析必须满足输入信号脉宽大于最小脉冲宽度的要求，否则时序分析结果就可能与实际情况不一致，主要原因之一就是无法保证时序单元实现正常的逻辑功能。

2.4 时序路径

时序路径 (timing path) 是指设计中数据信号传播过程中所经过的逻辑路径。每一条时序路径都存在与之对应的一个始发点和一个终止点，如图 2-12 所示。

时序分析中定义的始发点可以分为两种：组合逻辑单元的数据输入端口和时序单元的时钟输入端口，如图 2-13 所示。

时序分析中定义的终止点也可以分为两种：组合逻辑单元的数据输出端口和时序单元的数据输入端口，如图 2-14 所示。

时序路径根据始发点和终止点的不同可分为 4 种类型的时序路径：触发器到触发器

(register to register)、触发器到输出端 (register to output)、输入端到触发器 (input to register) 和输入端到输出端 (input to output), 如图 2-15 所示。

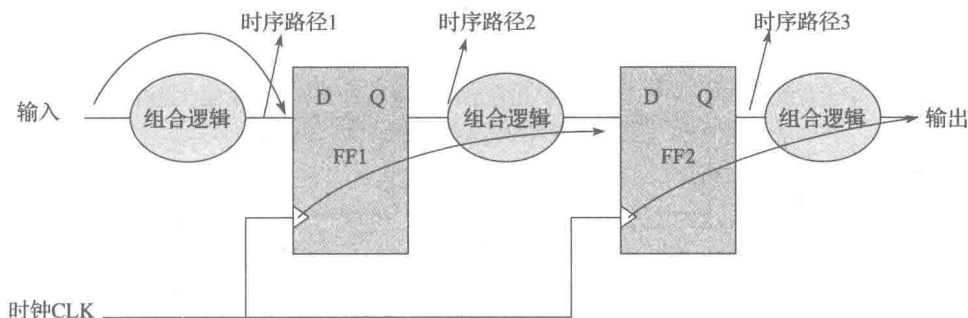


图 2-12 时序路径

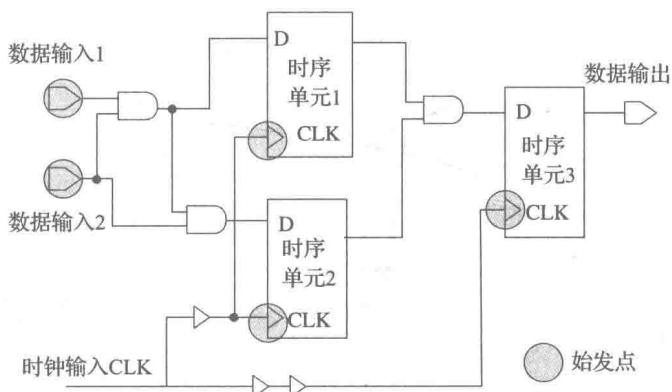


图 2-13 两种始发点

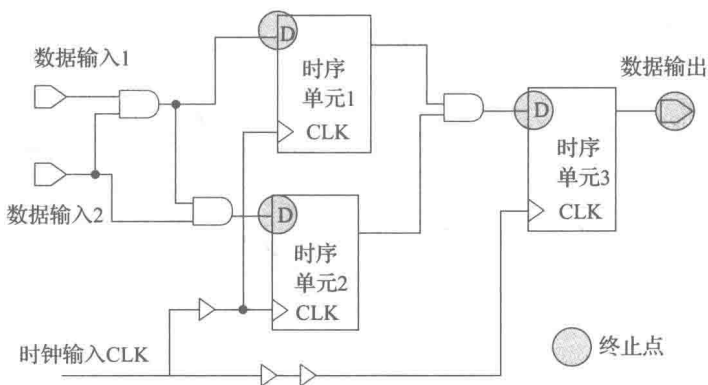


图 2-14 两种终止点

1. 触发器到触发器

这种类型的时序路径表示始发点为时序单元的时钟输入端和终止点为数据输入端之间的

时序路径。由于其始发点和终止点在设计内部都是可见的，所以也称为内部时序路径。

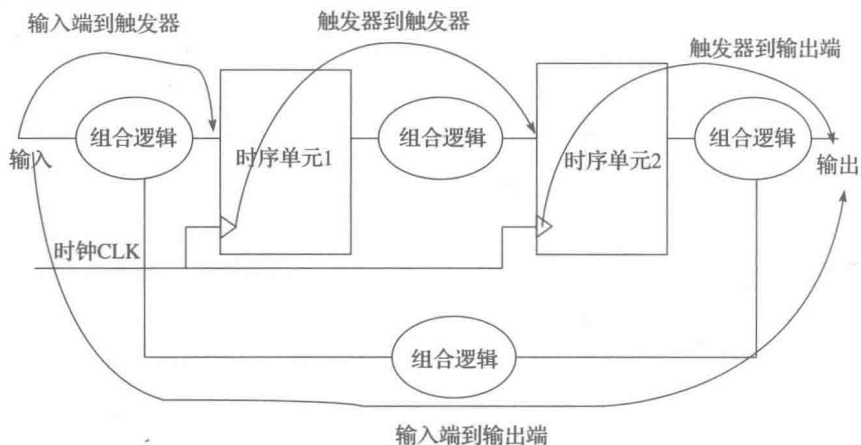


图 2-15 4 种时序路径

触发器到触发器的时序路径如图 2-16 所示。

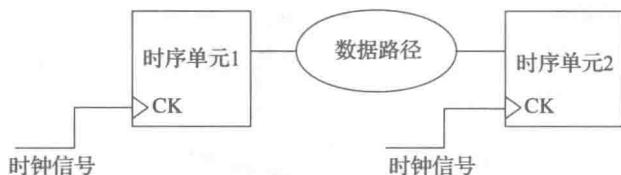


图 2-16 触发器到触发器时序路径

2. 触发器到输出端

这种类型的时序路径表示从始发点为时序单元时钟输入端口到终止点为组合逻辑单元的输出端口之间的时序路径。由于组合逻辑单元的输出端口可能连接到设计之外的其他模块的输入端口，所以称为外部时序路径。

触发器到输出端的时序路径如图 2-17 所示。

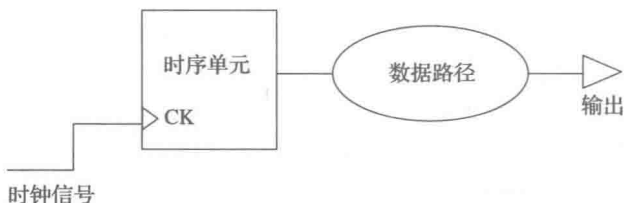


图 2-17 触发器到输出端时序路径

3. 输入端到触发器

这种类型的时序路径表示从始发点为组合逻辑单元的输入端口到终止点为时序单元数据

输入端之间的时序路径。由于组合逻辑单元的输入端口来自设计之外的其他输出端口，所以也称为外部时序路径。输入端到触发器的时序路径如图 2-18 所示。

4. 输入端到输出端

这种类型的时序路径表示从始发点为组合逻辑单元输入端口到终止点为组合逻辑单元输出端口之间的时序路径。因为没有经过任何时序单元，所以该种类型的时序路径比较特殊，其时序分析方法也不同。输入端到输出端的时序路径如图 2-19 所示。

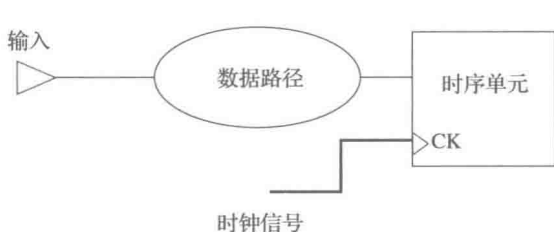


图 2-18 输入端到触发器时序路径

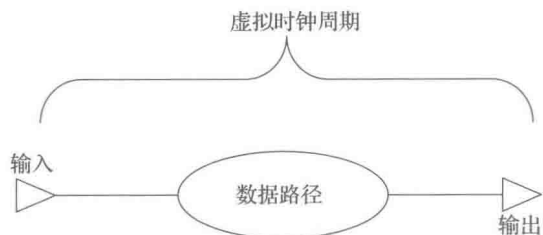


图 2-19 输入端到输出端时序路径

2.5 时钟特性

在同步电路设计中，各功能逻辑单元之间的数据传输都由一个同步信号控制，这个执行统一指挥的信号就是时钟信号。时钟信号为一个周期性信号，如图 2-20 所示。

随着设计工艺水平的不断提高，由于时钟本身所引起时序问题的现象越来越严重，因此有必要了解和掌握高速时钟本身的时序特性，并且在静态时序分析过程中要把这些影响设计时序性能的时钟特性考虑进来。

时钟的时序特性主要分为时钟周期（clock period）、时钟占空比（clock duty cycle）、时钟转换时间（clock transition time）、时钟延迟（clock latency）、时钟偏斜（clock skew）和时钟抖动（clock jitter）。

1. 时钟周期

时钟周期也称为振荡周期，定义为时钟频率的倒数。如图 2-21 所示为一个时钟周期为 10 的时钟信号波形。

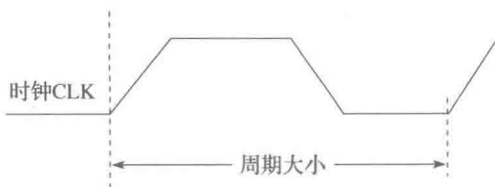


图 2-20 时钟信号

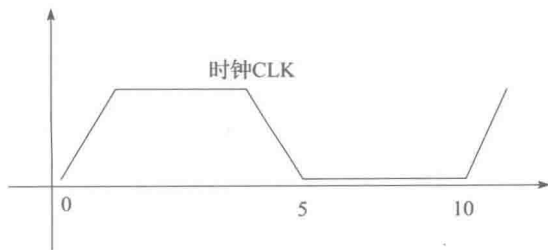


图 2-21 时钟信号波形

静态时序分析中需要设置时钟周期即时钟信号的工作频率，作为时序参考基础，来检查时序路径是否满足时序要求。

2. 时钟占空比

时钟占空比是指时钟信号高电平在一个周期之内所占的时间比率，如图 2-22 所示。

一般设计中，大部分的时钟占空比为 50%，即占空比为 0.5，说明高、低电平所占时间都为 0.5 个周期。但根据设计需要，也可以设计占空比不同的时钟信号，如图 2-23 所示为占空比为 0.333 的时钟信号波形。

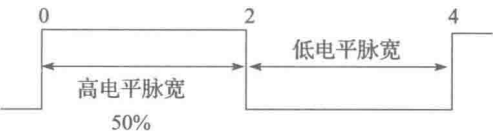


图 2-22 时钟占空比为 0.5

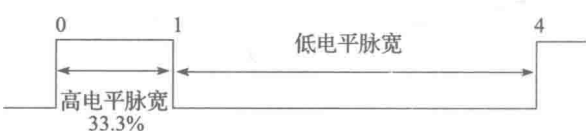


图 2-23 时钟占空比为 0.333

3. 时钟转换时间

理想情况下，时钟信号在不同电平之间进行切换是不需要时间的，但是实际时钟信号在不同电平之间切换时，由于供电电压、工艺变化、扇出负载大小和互连负载大小等因素，是需要转换时间的。因此，时钟转换时间一般定义为时钟信号的电压从标准供电电源电压的 10% 变化到标准供电电源电压的 90% 的时间间隔。根据设计的需求，设计者可以通过单元时序计算参数来定义不同的时钟转换时间参数，如图 2-24 所示。

由于时钟转换时间分为上升和下降两种情况，应该使上升和下降转换的时间保持对称性。基于时钟信号输出上升和下降转换的时间保持对称性的需要，时钟转换时间越短，而且不同切换形式下越对称，那么时钟信号质量越好。

4. 时钟延迟

时钟延迟是指时钟信号从时钟源输出端口到达时序单元时钟输入端口所需要的传播时间，如图 2-25 所示。

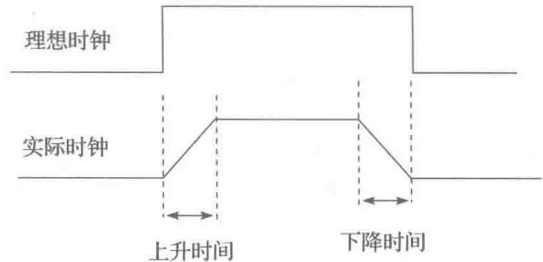


图 2-24 时钟转换时间

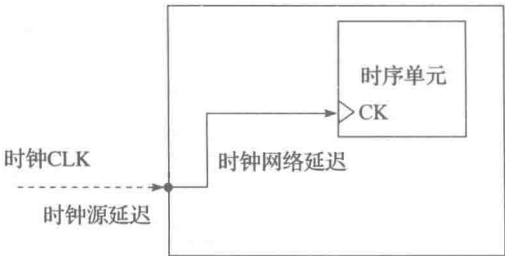


图 2-25 时钟延迟

由于 OCV (片上工艺偏差) 和 PVT (process (工艺)、voltage (电压)、temperature (温度)) 等因素会不同幅度地影响时钟输入的延时不确定性，从而导致整个设计时序的不确定。所以

时钟输入延时越短,时钟树性能越好,这样可以减少其他外在因素对时钟树性能的影响。

5. 时钟偏斜

非理想情况下,由于时钟线长度及时钟树叶节点负载不同等因素,导致时钟信号到达一时序路径下的相邻两个时序单元时钟端口的时间并不相同,这种时钟信号之间的偏移就是相同时钟信号之间的时钟偏斜,如图 2-26 所示。

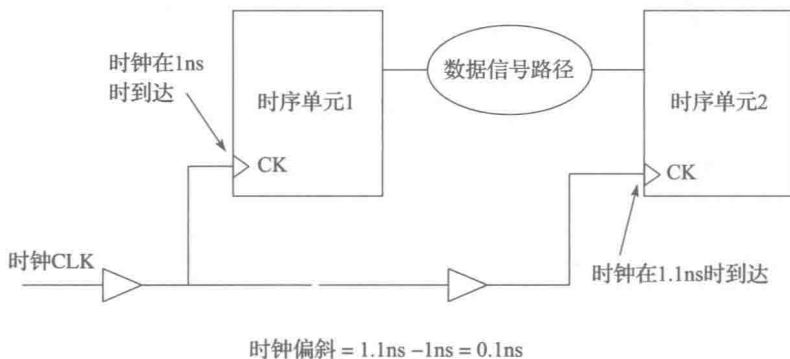


图 2-26 时钟偏斜

实际设计中,时钟信号到达每一个时序单元时钟端口的延时不可能完全相同,时钟偏斜是肯定存在的,这是时序分析中必须要考虑的因素。静态时序分析主要分为布局布线前和布局布线后两个阶段,两者的主要区别在于:后者有具体的互连线长度、宽度、信号分布情况等信息,所以后者可以更加准确地估计互连线延迟,以及时钟树网络的延迟;而前者只能根据设计电路和面积的大小等简单信息估计线上延迟和时钟树的延迟。

6. 时钟抖动

芯片工作过程中通常都是根据系统时钟脉冲信号同步工作的。最理想的情况是,时钟输入信号在下一个时钟的有效电平或者信号边缘到来之前切换,并在其正确的逻辑电平上保持稳定,从而使整个电路系统的行为合乎预设。但是在时钟电路设计实现中,不同实现单元的速度在不同时刻可能有着大小不一的差别,时钟信号可能并不能准确地在理想的信号边缘到来之前的瞬间保持在其正确的信号值上,它保持稳定所需的时间比理想情况有一定的偏移,这种偏移是在同一个时序单元的时钟输入端口上的时钟偏移。该时钟偏移主要表现为时钟抖动,如图 2-27 所示。

时钟抖动是永远存在的,该时钟抖动带来的影响在静态时序分析中可以通过设置时序裕度值来解决。

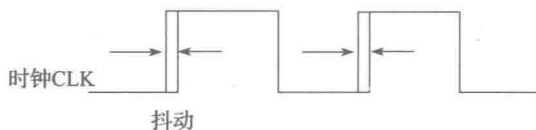


图 2-27 时钟抖动

2.6 时序弧

静态时序分析是基于时序弧 (timing arc) 数据的时序分析。时序弧是用来描述两个节点

延时信息的数据，时序弧的信息一般分为连线延时和单元延时。连线延时是单元输出端口和扇出网络负载之间的延时信息；单元延时是单元输入端口到输出端口的延时信息。因此一条完整路径的时序信息计算由连线延时和单元延时组成，如图 2-28 所示。

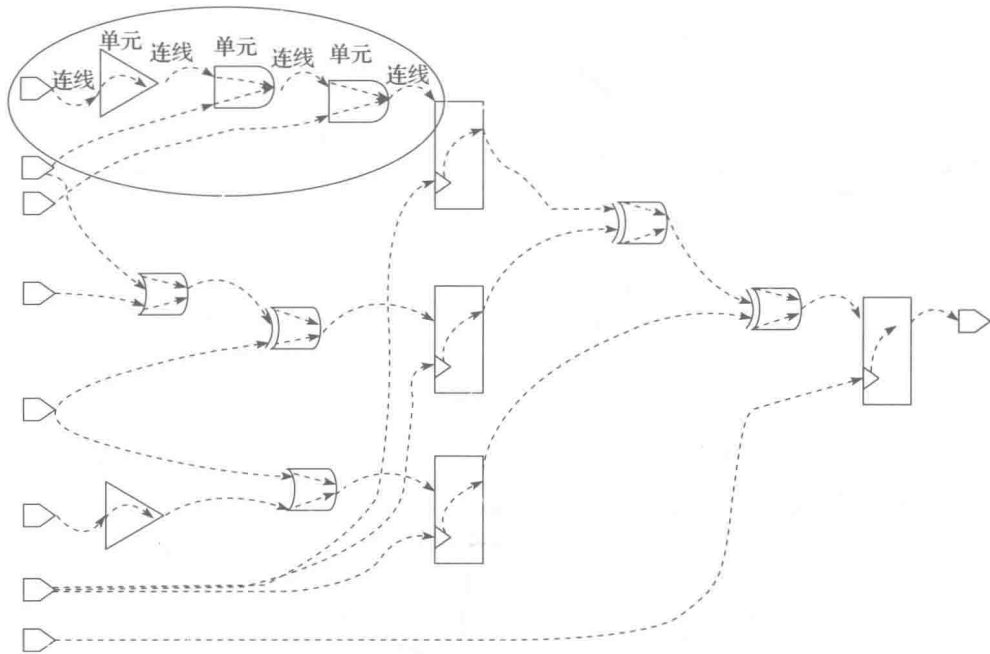


图 2-28 时序弧

因为连线延时没有功能属性，所以延时信息在分类上没有区别。单元延时中的时序弧分为基本时序弧和约束时序弧两类，其中约束时序弧用来表示输入端口之间存在的时序约束信息。

基本时序弧包括组合时序弧（combinational arc）、边沿时序弧（edge arc）、复位清零时序弧（preset and clear arc）、三态使能时序弧（three state enable and disable arc）等。

约束时序弧包括建立时序弧（setup arc）、保持时序弧（hold arc）、恢复时序弧（recovery arc）、移除时序弧（removal arc）和脉宽时序弧（width arc）等。

接下来将对几种比较常用的时序弧进行介绍。

1. 组合时序弧

组合时序弧是最基本的单元延时信息时序弧，主要用于表示组合逻辑单元的延时信息，在时序信息文件中通过时序弧类型信息 Combinational 进行声明，如下所示。

```
timing_type : combinational;
```

由于组合时序弧也是默认的时序弧类型，因此没有特别声明的时序弧就默认为组合时序弧。

组合时序弧根据单元功能逻辑的不同,分为基于3种不同逻辑状态下的延时弧,这3种逻辑状态的延时弧分别为同向时序弧(positive unate arc)、反向时序弧(negative unate arc)和不定态时序弧(non-unate arc)。

1) 同向时序弧表示输入端信号变化方向与输出端信号变化方向一致的时序弧,如图2-29所示。

同向时序弧在时序信息文件中通过定义相关PIN和逻辑状态信息positive_unate进行声明,如下所示。

```
related_pin : "A";
timing_sense : positive_unate;
```

2) 反向时序弧表示输入端信号变化方向与输出端信号变化方向相反的时序弧,如图2-30所示。

反向时序弧在时序信息文件中通过定义相关PIN和逻辑状态信息negative_unate进行声明,如下所示。

```
related_pin : "A";
timing_sense : negative_unate;
```

3) 不定态时序弧表示输入信号变化方向与输出端信号变化方向无固定确定关系的时序弧,如图2-31所示。

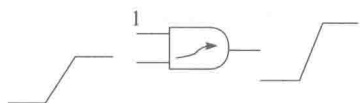


图 2-29 同向时序弧

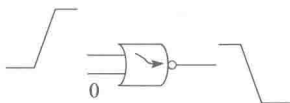


图 2-30 反向时序弧

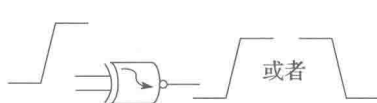


图 2-31 不定态时序弧

不定态时序弧在时序信息文件中通过定义相关PIN和逻辑状态信息non_unate进行声明,如下所示。

```
related_pin : "A";
timing_sense : non_unate;
```

需要强调的是,其他类型的时序弧如果也存在不同的逻辑状态,也需要通过使用以上3种逻辑状态信息进行声明。

2. 边沿时序弧

边沿时序弧用来表示时序单元从时钟信号有效沿到输出信号有效的延时信息。

如图2-32所示,从时序单元时钟高电平有效沿开始到时序单元QB输出端信号有效的延时时即通过边沿时序弧表示。

边沿时序弧根据时钟信号有效沿不同,分为上升沿有效和下降沿有效两种,所以在时序信息文件中分别通过时序弧类型信息falling_edge和rising_edge进行声明,如下所示。

```
timing_type : falling_edge;  
timing_type : rising_edge;
```

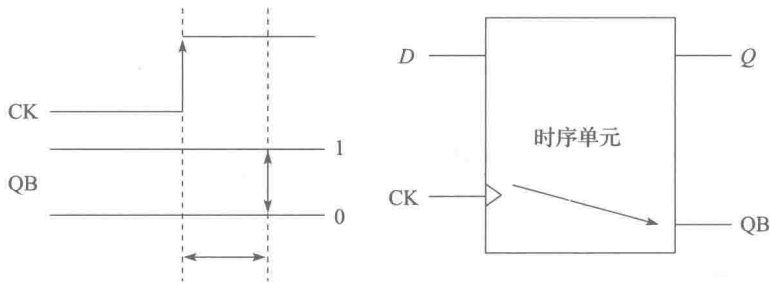


图 2-32 边沿时序弧

3. 复位清零时序弧

复位清零时序弧用来表示具有复位清零端口的时序单元从复位清零信号有效沿到输出信号有效的延时信息。

如图 2-33 所示，从时序单元清零高电平有效沿开始到时序单元 QB 输出端信号有效的延时时即通过复位清零时序弧表示。

复位清零时序弧根据复位清零信号有效沿的不同，分为上升沿有效和下降沿有效两种，所以在时序信息文件中分别通过时序弧类型信息 preset 和 clear 进行声明，如下所示。

```
timing_type : preset;  
timing_type : clear;
```

4. 三态使能时序弧

三态使能时序弧用来表示三态逻辑单元从使能信号有效到输出的延时信息。

如图 2-34 所示，从三态逻辑单元使能信号有效到三态输出的延时时即通过三态使能时序弧表示。

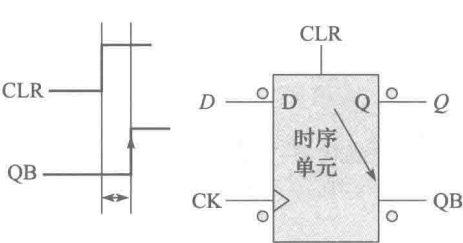


图 2-33 复位清零时序弧

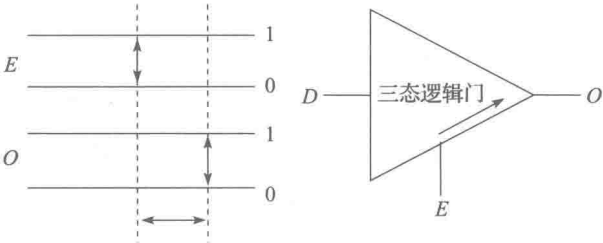


图 2-34 三态使能时序弧

三态使能时序弧中根据使能信号位有效与无效的不同，在时序信息文件中分别通过状态信息 three_state_enable 和 three_state_disable 进行声明，如下所示。

```
timing_type : three_state_enable;  
timing_type : three_state_disable;
```

5. 建立时序弧

建立时序弧用来表示时序单元获得正确数据信号所需要的约束信息，即在时钟沿有效前数据输入信号保持有效的时间信息。

如图 2-35 所示，在时序单元时钟高电平有效沿之前，信号必须已经有效的时间信息即通过建立时序弧表示。

建立时序弧根据时钟信号有效沿的不同，分为上升沿有效和下降沿有效两种，所以在时序信息文件中分别通过时序弧类型信息 `setup_rising` 和 `setup_falling` 进行声明，如下所示。

```
timing_type : setup_rising;
timing_type : setup_falling;
```

6. 保持时序弧

保持时序弧也是用来表示时序单元获得正确信号所需要的约束信息，即在时钟沿有效后数据输入信号继续维持有效的时间信息。

如图 2-36 所示，在时序单元时钟高电平有效沿之后，数据信号必须继续维持有效的的时间信息通过保持时序弧表示。

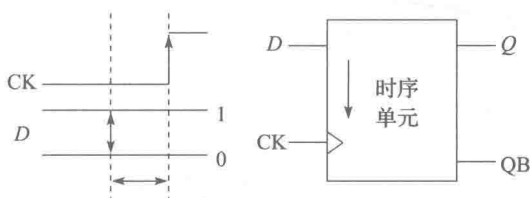


图 2-35 建立时序弧

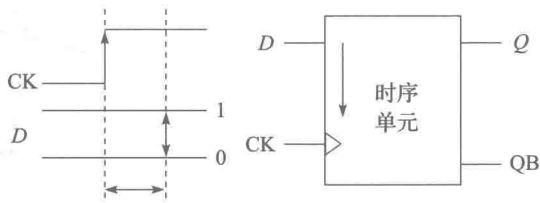


图 2-36 保持时序弧

保持时序弧根据时钟信号有效沿的不同，分为上升沿有效和下降沿有效两种，所以在时序信息文件中分别通过时序弧类型信息 `hold_rising` 和 `hold_falling` 进行声明，如下所示。

```
timing_type : hold_rising;
timing_type : hold_falling;
```

7. 恢复时序弧

恢复时序弧用来表示具有复位清零端口的时序单元成功使能所需要的约束信息，即在时钟沿有效前使能信号保持有效的的时间信息。

如图 2-37 所示，在时序单元时钟高电平有效沿之前，清零信号必须提前有效的的时间信息即通过恢复时序弧表示。

恢复时序弧根据时钟信号有效沿的不同，分为上升沿有效和下降沿有效两种，所以在时序信息文件中分别通过时序弧类型信息 `recovery_rising` 和 `recovery_falling` 进行声明，如下所示。

```
timing_type : recovery_rising
timing_type : recovery_falling
```

8. 移除时序弧

移除时序弧也是用来表示具有复位清零端口的时序单元成功使能所需要的约束信息，即在时钟沿有效后使能信号维持有效的时间信息。

如图 2-38 所示，在时序单元时钟高电平有效沿之后，清零信号必须继续维持有效的时
间信息通过恢复时序弧表示。

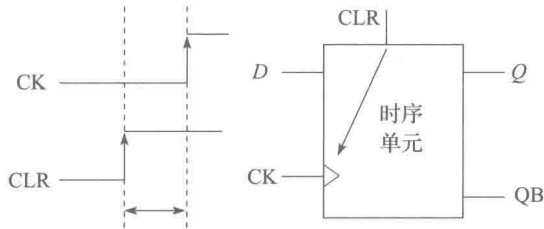


图 2-37 恢复时序弧

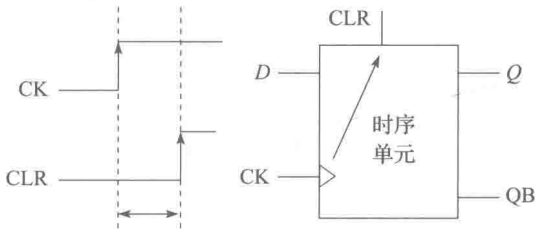


图 2-38 移除时序弧

移除时序弧根据时钟信号有效沿的不同，分为上升沿有效和下降沿有效两种，所以在
时序信息文件中分别通过时序弧类型信息 `removal_rising` 和 `removal_falling` 进行声明，如下
所示。

```
timing_type : removal_rising
timing_type : removal_falling
```

9. 脉宽时序弧

脉宽时序弧也是用来表示时序单元获得正确信号所需要的约束信息，即时钟信号同一状
态下维持有效的最小总时间信息，如图 2-39 所示。

脉宽时序弧根据信号有效状态的不同，
分为高电平有效和低电平有效两种，所以在
时序信息文件中通过时序弧类型信息 `min_`
`pulse_width` 进行声明，如下所示。

```
timing_type : min_pulse_width
```

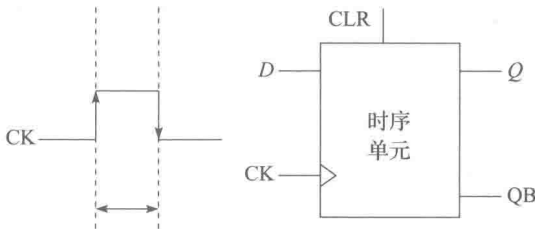


图 2-39 脉宽时序弧

2.7 PVT 环境

为了使静态时序分析的结果更接近实际芯片工作环境的情况，需要对芯片设计工作环
境，包括工艺、工作电压、工作温度等参数进行设置，该参数的组合简称为 PVT。

PVT 各参数都可以分为典型的 (typical)、最好的 (best) 和最坏的 (worst) 3 种工艺情况。
对应不同的工艺情况，单元延时也不一样，如图 2-40 所示。

由图 2-40 可以看出，best 工艺模型下单元延时最快，worst 模型下单元延时最慢，
typical 模型下单元延时处于中等水平。

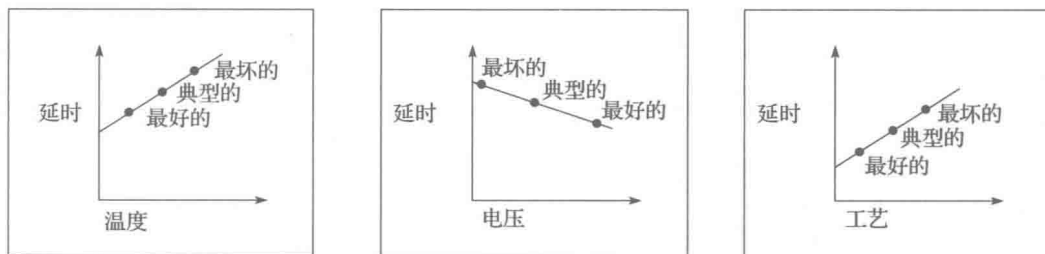


图 2-40 PVT 环境

根据不同的时序分析需要，有不同的 PVT 组合条件可选择。以下为 3 种常规的 STA 分析条件。

1. TYP (Typical)

TYP 工艺条件包括典型的工艺 (typical process)、典型的工艺温度 (nominal temperature) 和典型的有效电源电压 (nominal voltage)。典型的工艺温度即为常温 25℃，典型的有效电源电压即为工艺定义的标准供电电压。

以下为时序信息文件中对 TYP 工艺条件的定义内容示例：

```
/* operation conditions */
nom_process : 1;
nom_temperature : 25;
nom_voltage : 1.2;
operating_conditions(typ) {
    process : 1;
    temperature : 25;
    voltage : 1.2;
    tree_type : balanced_tree
}
default_operating_conditions : typ;
```

以上示例定义的 TYP 工艺条件名称为 typ，其中温度定义为 25℃，电压定义为 1.2V。

2. BCF (Best-Case Fast)

BCF 工艺条件包括最快的工艺 (fast process)、最低的工艺温度 (lowest temperature) 和最高的有效电源电压 (highest voltage)。与 WCS 相同，在先进工艺，比如 28nm 下，单元延时大小由于在温度参数上的反转，BCF 工艺条件包括最快的工艺 (fast process)，最高工艺温度 (highest temperature) 和最高的有效电源电压 (highest voltage)。最低的工艺温度根据不同的制造工艺决定，一般为 -40℃。最高的有效电源电压大小一般为典型有效电源电压的 110%。

如下所示为时序信息文件中对 BCF 工艺条件的定义内容示例：

```
/* operation conditions */
nom_process : 1;
nom_temperature : -40;
```



```

nom_voltage : 1.32;
operating_conditions(bcf) {
    process : 1;
    temperature : -40;
    voltage : 1.32;
    tree_type : balanced_tree
}
default_operating_conditions : bcf;

```

以上示例定义 BCF 工艺条件名称为 bcf，其中温度定义为 -40℃，电压定义为 1.32V。

3. WCS (Worst-Case Slow)

通常 WCS 工艺条件包括最慢的工艺 (slow process)、最高的工艺温度 (highest temperature) 和最低的有效电源电压 (lowest voltage)。但是随着制造工艺水平的不断提高，比如 28nm 制造工艺下，单元延时随温度降低延时变大，所以 WCS 工艺在温度参数上进行了反转，这样 WCS 工艺条件包括最慢的工艺 (slow process)、最低的工艺温度 (lowest temperature) 和最低的有效电源电压 (lowest voltage)。最高的工艺温度由不同的制造工作决定，一般为 125℃。最低的有效电源电压一般为典型有效电源电压的 90%。

以下为时序信息文件中对 WCS 工艺条件的定义内容示例：

```

/* operation conditions */
nom_process : 1;
nom_temperature : 125;
nom_voltage : 1.08;
operating_conditions(wcs) {
    process : 1;
    temperature : 125;
    voltage : 1.08;
    tree_type : balanced_tree
}
default_operating_conditions : wcs;

```

以上示例定义 WCS 工艺条件名称为 wcs，其中温度定义为 125℃，电压定义为 1.08V。

如果时序分析同时还要考虑功耗的分析，那么加入如下两种工艺情况。

(1) ML (maximal leakage)

ML 工艺条件包括最快的工艺 (fast process)、最高的工艺温度 (highest temperature) 和最高的有效电源电压 (highest voltage)。

以下为时序信息文件中对 ML 工艺条件的定义内容示例：

```

/* operation conditions */
nom_process : 1;
nom_temperature : 125;
nom_voltage : 1.32;
operating_conditions(ml) {
    process : 1;

```

```

temperature : 125;
voltage : 1.32;
tree_type : balanced_tree
}
default_operating_conditions : ml;

```

以上示例定义 ML 工艺条件名称为 ml，其中温度定义为 125℃，电压定义为 1.32V。

(2) TL (typical leakage)

TL 工艺条件包括典型的工艺 (typical process)、最高的工艺温度 (highest temperature) 和典型的有效电源电压 (nominal voltage)。该工艺下工艺温度选择最高温度而不是典型的温度，这是因为根据具体的实际情况，芯片在消耗功耗的时候其温度是高于常温的。

以下为时序信息文件中对 TL 工艺条件的定义内容示例：

```

/* operation conditions */
nom_process : 1;
nom_temperature : 125;
nom_voltage : 1.2;
operating_conditions(tl) {
    process : 1;
    temperature : 125;
    voltage : 1.2;
    tree_type : balanced_tree
}
default_operating_conditions : tl;

```

以上示例定义 TL 工艺条件名称为 tl，其中温度定义为 125℃，电压定义为 1.2V。

参数 nom_process 为工艺缩放因子，该参数值根据工艺厂家的生产流片情况而定，其值越小则时序分析的延时结果越小，反之延时结果越大。

参数 tree_type 用于定义时序分析中使用的互连线延时计算模型来计算信号线传播延时。互连线延时计算模型分为 3 种：best_case_tree、worst_case_tree 和 balanced_tree。这 3 种互连线延时计算模型的区别如图 2-41 所示。

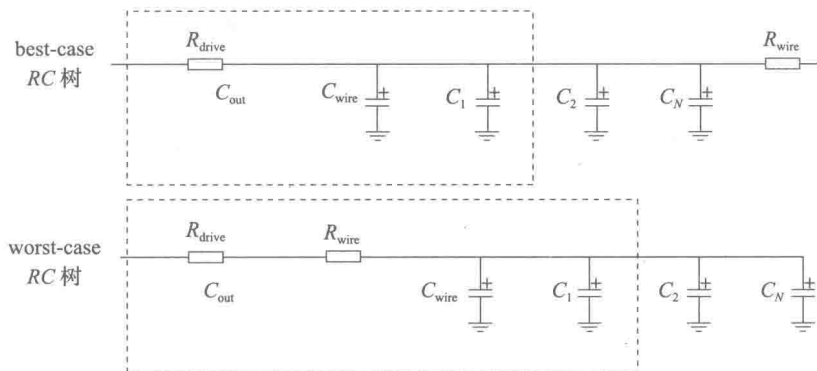


图 2-41 3 种互连线延时计算模型

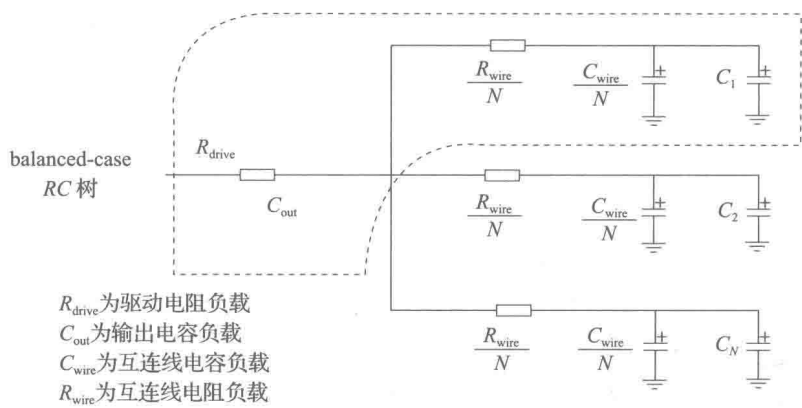


图 2-41 (续)

如图 2-41 所示，互连线延时计算模型 `best_case_tree` 只考虑驱动电阻负载、互连线负载总电容值和对应驱动负载电容值；互连线延时计算模型 `worst_case_tree` 相比 `best_case_tree` 还多考虑互连线总电阻负载；互连线延时计算模型 `balanced_tree` 相比 `worst_case_tree` 把互连线总电阻、电容负载根据负载节点的个数多少进行了平均分配。

在时序分析中，通过命令 `set_operating_conditions` 来定义需要使用的工艺环境，如下所示。

```
set_operating_conditions typ -library TYP
```

如上命令定义当前使用的工艺环境为典型的工艺环境。

2.8 时序计算单位

时序分析中需要基于温度、电压、电流、电阻、电容负载和时间等不同类型的变量因子进行计算，因此需要在时序信息文件中定义不同类型的变量因子的单位。以下为时序信息文件中所定义的变量因子单位：

```
/* unit attributes */  
time_unit : "1ns";  
voltage_unit : "1V";  
current_unit : "1mA";  
pulling_resistance_unit : "1kohm";  
leakage_power_unit : "1pW";  
capacitive_load_unit (1.0,pF);
```

以上示例定义时间单位为 1ns，电压单位为 1V，电流单位为 1mA，电阻负载单位为 1kΩ，亚阈漏功耗单位为 1pW，电容负载单位为 1pF。根据设计的需求，设计者可以在时序信息文件中设置不同的变量单位。

第 3 章

单元库时序模型

在时序分析过程中, 对一个复杂芯片中的每一个模块, 不论是简单的标准单元 (如 NAND、NOR 等), 还是复杂的定制设计模块 (如 RAM、处理器核等), 都需要有一个时序模型。

3.1 基本时序模型简介

针对模块不同的规模和特点, 通常分为以下 5 种时序模型。

1. 快速时序模型

在设计过程中, 可以对一些未完成的关键模块预先建立快速时序模型 (quick time model), 其中包含该模块粗略的时序描述。在设计周期的早期, 利用快速时序模型可以对整个设计的时序性能进行预估, 便于随时调整设计以满足性能要求。在设计后期, 各模块已设计完成, 便可用详细的时序模型代替快速时序模型, 以得到精确的时序信息。

快速时序模型建模方法如下。

通过时序信息库对应单元的时序弧信息, 把需要建模的时序路径的时序弧近似为某些单元的时序弧组合。

对于含有时序单元和组合逻辑单元组合的快速时序模型建模方法, 式 (3-1) 如下所示:

$$\begin{aligned} \text{快速时序模型建模延时} = & \text{单元级数} \times \text{单元延时} + \text{时序单元 CLK-Q 延时} \\ & + \text{带负载的输出延时} \end{aligned} \quad (3-1)$$

对于只含有组合逻辑单元组合的快速时序模型建模方法, 式 (3-2) 如下所示。

$$\text{快速时序模型建模延时} = \text{单元级数} \times \text{单元延时} + \text{带负载的输出延时} \quad (3-2)$$

同样, 定义需要建模的时序路径的约束时序弧近似为某些时序单元的约束时序弧, 式 (3-3) 如下所示。

$$\text{快速时序模型建模延时} = \text{单元级数} \times \text{单元延时} + \text{时序单元建立时间} \quad (3-3)$$

快速时序模型建模的电路图示例如图 3-1 所示。

根据图 3-1 所示的电路图, 通过前面介绍的快速时序模型建模方法, 在基于 CLK 约束的端口 A 的快速时序模型建模中的建立时间 SetupA 如式 (3-4) 所示。

$$\text{SetupA} = \text{COMB1 的延时} + \text{时序单元 DFF1 的建立时间} \quad (3-4)$$

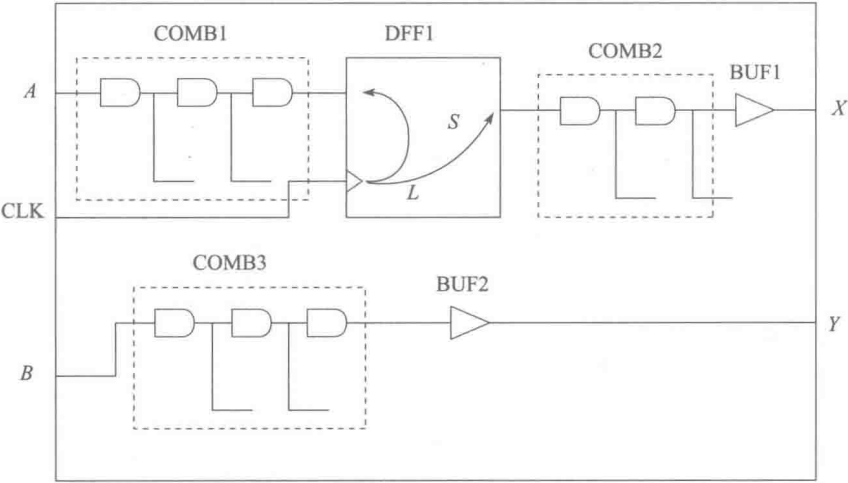


图 3-1 快速时序模型建模的电路图

其中，COMB1 的延时 = 3×2 输入与门的延时。

基于 CLK 约束的端口 A 的快速时序模型建模中的保持时间 HoldA 如式 (3-5) 所示。

$$\text{HoldA} = \text{COMB1 的延时} + \text{时序单元 DFF1 的保持时间} \quad (3-5)$$

其中，COMB1 的延时 = 3×2 输入与门的延时。

端口 CLK 到端口 X 的快速时序模型建模中的延时 CLKtoX 如式 (3-6) 所示。

$$\text{CLKtoX} = \text{DFF1 的 CLK-Q 延时} + \text{COMB2 的延时} + \text{单元 BUF1 的延时} \quad (3-6)$$

其中，COMB2 的延时 = 2×2 输入与门的延时。

端口 B 到端口 Y 的快速时序模型建模中的延时 BtoY 如式 (3-7) 所示。

$$\text{BtoY} = \text{COMB3 的延时} + \text{单元 BUF2 的延时} \quad (3-7)$$

其中，COMB3 的延时 = 3×2 输入与门的延时。

最终，图 3-1 所示的电路图在快速时序模型建模后的时序弧信息图如图 3-2 所示。

2. 接口逻辑模型

接口逻辑模型 (interface logic model) 用一种结构化的方法，将原始电路简化为只包含接口逻辑的一个小电路，如图 3-3 所示。

一个模块的接口逻辑只包含影响其他模块时序的单元以及被其他模块影响的单元逻辑，而不包括内部触发器到触发器的路径逻辑信息。

3. 抽取时序模型

抽取时序模型 (extracted timing model) 从一个模块的门级网表中抽取出模块的时序信息，可以隐藏单元的内部实现细节，有利于保护知识产权。如图 3-4 所示为该设计

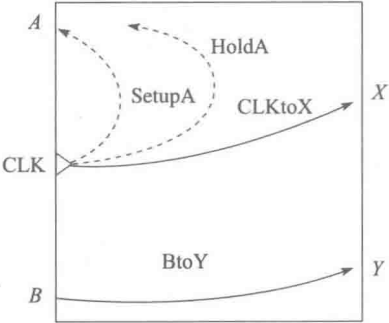


图 3-2 快速时序模型建模后的时序弧信息图

的电路图。

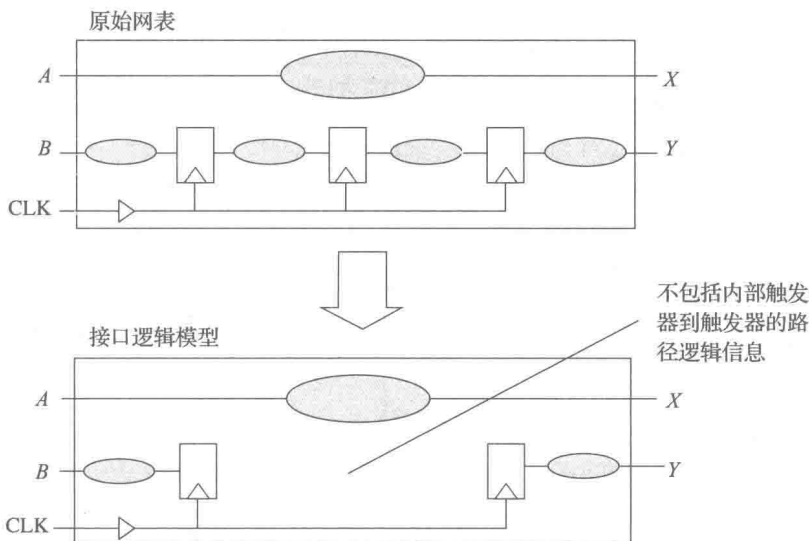


图 3-3 接口逻辑模型简化

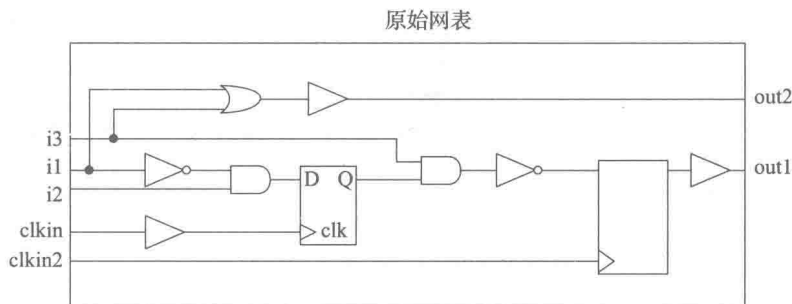


图 3-4 抽取时序模型前电路图

抽取时序模型表达后，内部电路的实现细节被很好地隐藏了，如图 3-5 所示。抽取时序模型的时序弧信息如图 3-6 所示。抽取时序模型表达后，内部时序弧实现细节被很好地隐藏了。

4. Stamp 模型

对于一个全定制设计的模块（如 SRAM），将其作为一个固定的单元放入芯片，如果此类模块没有详细的门级网表，那么用 Synopsys 公司的静态时序分析工具 PrimeTime 提供的一种 Stamp 模型描述语言可以对此类模块的时序行为进行描述。Stamp 模型目前基本过时了，已经被其他类型的时序模型所代替，因此这里不再深入介绍。

5. Synopsys 工艺库模型

基于标准单元的半定制设计中，综合工具将 RTL 级设计描述转换成门级网表，每个标准单元均需要一个时序模型描述，才能对其进行时序分析。标准单元性能参数文档图示例如

图 3-7 所示。

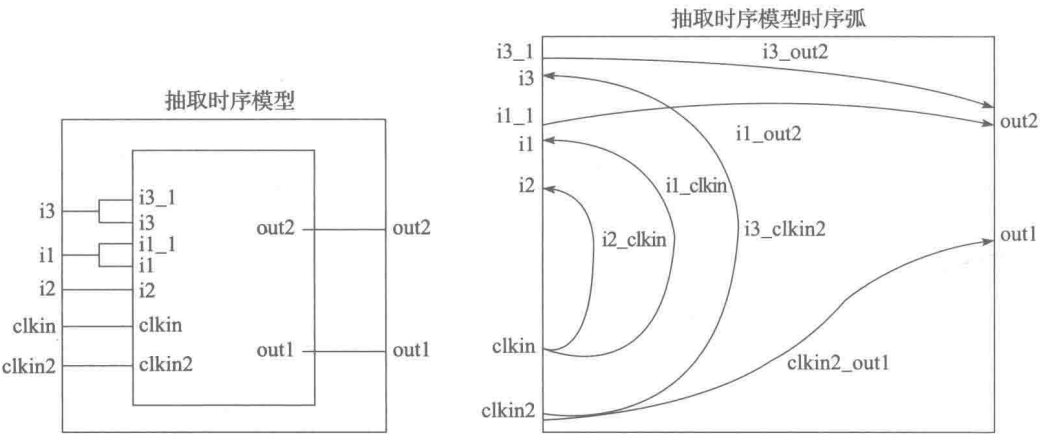


图 3-5 抽取时序模型后的电路

图 3-6 抽取时序模型的时序弧信息

小驱动力的与门名字

单元功能文字描述

单元MOS晶体管数量和IO PAD数量

| CELL NAME | FUNCTION | CELL COUNT | |
|-----------|-------------|------------|-----|
| | | GATE | I/O |
| AN2 | 2-INPUT AND | 2 | 0 |
| AN2P | | 2 | 0 |

大驱动力的与门名字

LOGIC SYMBOL 单元电路符号图

TRUTH TABLE 单元功能真值表

| INPUT | | OUTPUT |
|-------|---|--------|
| A | B | Z |
| L | L | L |
| L | H | L |
| H | L | L |
| H | H | H |

上拉本征延时

时序弧路径

上拉时电阻负载值

下拉本征延时

下拉时电阻负载值

| CELL | PATH | TpLH | | TdLH | |
|------|---------|------|-------|------|-------|
| | | Tup | Kup | Tdn | Kdn |
| AN2 | A, B->Z | 0.39 | 0.122 | 0.53 | 0.038 |
| AN2P | A, B->Z | 0.51 | 0.053 | 0.58 | 0.023 |
| UNIT | | nS | nS/LU | nS | nS/LU |

输入引脚电容负载值

相关引脚信息

输出驱动力信息

| INPUT LOAD (LU) | |
|-----------------|----------|
| CELL | PIN A, B |
| AN2 | 1.0 |
| AN2P | 1.0 |

电容负载值

| OUTPUT DRIVE (LU) | |
|-------------------|-------|
| CELL | PIN Z |
| AN2 | 25 |
| AN2P | 57 |

输出最大扇出值

图 3-7 与门单元性能参数文档

目前,大部分标准单元库时序信息均采用 Synopsys 标准建模语言进行描述, Synopsys 工艺库时序模型已经成为业界的库标准,几乎所有的时序分析工具都提供了到 Synopsys 库的直接转换。Synopsys 工艺库时序模型中包含对一个单元功能、时序以及其他一些参数如面积、功耗和设计规则的描述。Synopsys 工艺库模型文件的部分内容如下所示。

```
cell(PIC) {
    area : 14070.000;
    pad_cell : TRUE;
        dont_touch : TRUE;
        dont_use : TRUE;
    pin(Y) {
        direction : output;
        max_capacitance : 0.417;
        capacitance : 0.0;
        function : "P*IE";
        timing() {
            related_pin : "IE" ;
            timing_sense : positive_unate;
            cell_rise(core_core_template_2d) {
                index_1 ("0.003, 0.669, 1.335, 2.002");
                index_2 ("0.050, 0.142, 0.233, 0.325, 0.417");
                values ( \
                    "0.790, 0.806, 0.819, 0.831, 0.842", \
                    "1.042, 1.058, 1.071, 1.083, 1.095", \
                    "1.263, 1.279, 1.292, 1.304, 1.316", \
                    "1.467, 1.483, 1.496, 1.508, 1.520" )}
            }
        internal_power() {
            related_pin : "IE";
            rise_power(core_core_energy_template_2d) {
                index_1 ("0.003, 0.669, 1.335, 2.002");
                index_2 ("0.050, 0.142, 0.233, 0.325, 0.417");
                values ( \
                    "3.382, 3.441, 3.485, 3.518, 3.551", \
                    "3.829, 3.888, 3.928, 3.964, 3.994", \
                    "4.260, 4.319, 4.358, 4.394, 4.426", \
                    "4.641, 4.700, 4.745, 4.779, 4.810" )}
            }
        }
    }
}
```

由以上示例可以看出, Synopsys 工艺库模型文件通常也可称为时序信息文件 (.lib)。前面介绍的时序模型的实现都是基于 Synopsys 工艺库模型的,因此后面章节都基于 Synopsys 工艺库模型进行介绍。

3.2 Synopsys 工艺库模型

前面已经介绍过,目前的大部分标准单元库时序信息均采用 Synopsys 标准建模语言进

行描述, Synopsys 工艺库时序模型已经成为业界的库标准, 几乎所有的时序分析工具都提供了到 Synopsys 库的直接转换。根据应用和计算精度的不同, 目前常用的 Synopsys 工艺库模型主要分为以下 3 种。

1. 线性延时模型

线性延时模型 (linear delay model) 是一种简单的 Synopsys 工艺库时序模型, 该时序模型定义单元的门延时和输出端转换延时可以基于输入端口转换延时和输出负载这两个变量的线性方程式来表达。但是在真实电路中, 门延时与输入信号跳变时间及输出负载之间并不是遵循严格的线性关系, 因此该延时计算模型不适合于对精度要求较高的延时计算。

2. 非线性延时模型

Synopsys 工艺库时序模型中, 目前业界主要使用非线性延时模型 (non-linear delay model), 它能够精确地表达逻辑门输出端和输入端的延时。它由驱动模型和接收模型组成, 如图 3-8 所示。

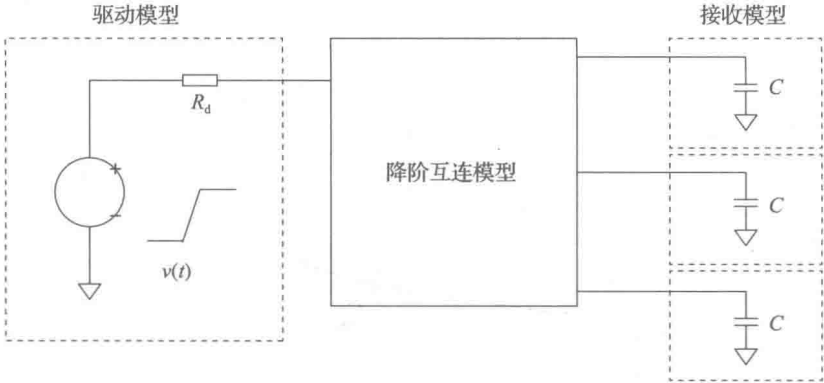


图 3-8 非线性延时模型的驱动模型和接收模型

由图 3-8 所示, 非线性延时模型的驱动模型为戴维南等效电压源。在 $0.13\mu\text{m}$ 工艺及以上时, 互连线电阻远小于驱动电阻, 逻辑门的输出波形基本可以通过戴维南等效电压源的计算结果来精确表达。非线性延时模型的接收模型即为单一指定的接收端电容负载。

3. 复合电流源延时模型

随着集成电路生产工艺水平的不断提高, 在 $0.13\mu\text{m}$ 工艺以下, 互连线电阻开始超过逻辑门的电阻大小, 成为决定时序延时的主要影响因子, 如图 3-9 所示。

根据非线性延时模型的驱动模型的戴维南等效电压源公式如式 (3-8) 所示。

$$V_{\text{out}} = V_{\text{in}} \times (Z_{\text{net}} / (Z_{\text{net}} + R_d)) \tag{3-8}$$

当 Z_{net} 远大于逻辑门电阻 R_d 时, 那么 V_{out} 基本等于 V_{in} , 这样的结果不再符合实际驱动端的状态。同时, 在 $0.13\mu\text{m}$ 工艺以下, 由于逻辑门中晶体管的米勒效应的影响开始变大, 其逻辑门输出波形不再像以前一样保持线性, 逻辑门输出波形的非线性导致前面介绍的非线性延时模型无法满足对逻辑门的输出波形进行精确建模。因此在 $0.13\mu\text{m}$ 工艺以下, 采用复

合电流源延时模型 (composite current source delay model, CCS 延时模型) 来进行时序分析计算。复合电流源延时模型也是由驱动模型和接收模型组成的, 如图 3-10 所示。

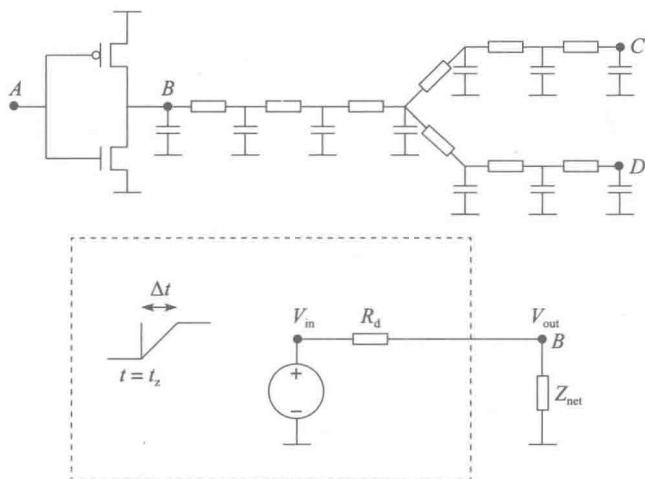


图 3-9 互连线电阻模型

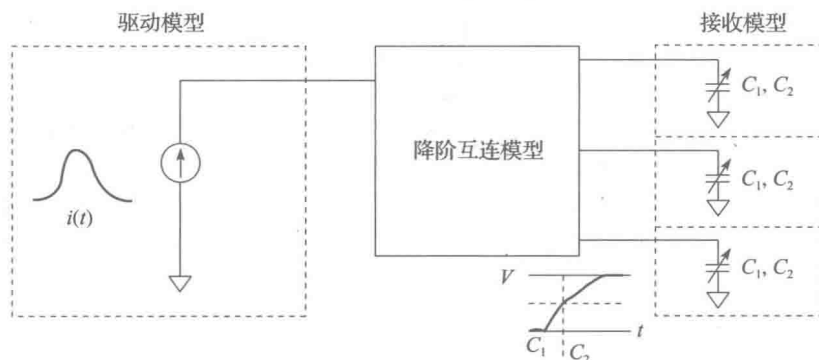


图 3-10 复合电流源延时模型的驱动模型和接收模型

如图 3-10 所示, 复合电流源延时模型的驱动模型是一个基于时间和电压的非线性电流源 $I_{out}(V_{out})(t)$, 同时设置电流源的驱动电阻为无穷大, 因此即使逻辑门的驱动电阻远低于互连线电阻, 该模型仍然能够提供很高的计算精确度。同时该模型达到高精度的另一个方面是不通过对晶体管的行为进行建模, 而是对随机情况下晶体管在不同的集总负载下的行为进行取样, 如图 3-11 所示。

如图 3-11 所示, a 为不同负载值下晶体管电流状态的模拟结果, b 为不同负载值下对应电压状态的模拟结果。通过对应不同的输入转换状态和输出负载状态, 复合电流源延时模型储存了电流数值, 并采用了一种能够减少电流、时间点数量的方法, 来精确地推导出逻辑门的输出引脚响应。

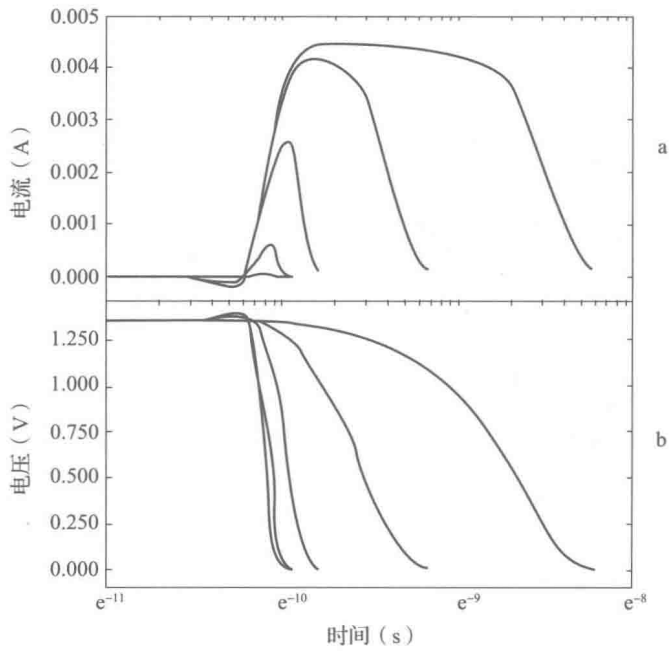


图 3-11 晶体管建模中电压与电流关系

由于在 $0.13\mu\text{m}$ 工艺以下，逻辑门中晶体管的米勒效应的影响开始变大，其逻辑门输出波形不再像以前一样保持线性，如果使用与非线性延时模型一致的接收模型，该接收模型为单一指定的接收端电容负载，则无法精确地反映实际的晶体管输出波形状态，如图 3-12 所示。

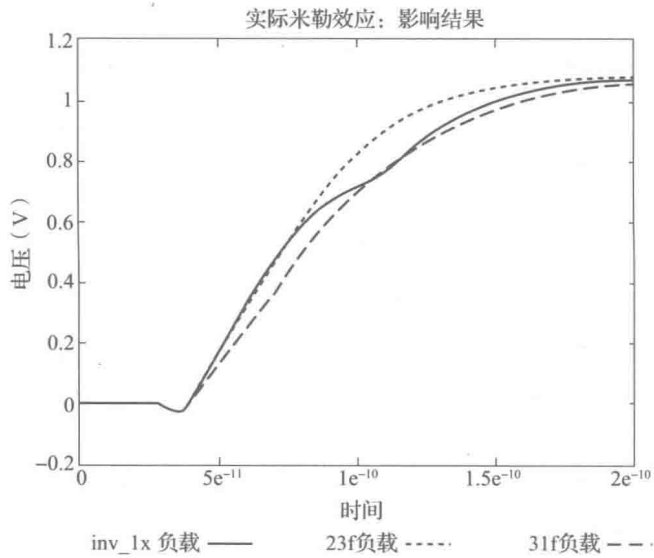


图 3-12 实际的晶体管输出波形状态

其输出负载值在某个时间点由于米勒效应的影响发生了突变,这样,基于单一负载值的输出波形无法精确反映该状态,因此复合电流源延时模型的接收模型由两个电容负载构成。复合电流源延时模型时序信息在提取时,第一个电容值由输出端电压的阈值所决定,当逻辑门的输出电压达到这个阈值时,其负载动态地调整到第二个值,并分别把电流、延时和电容负载等信息记录下来。由于复合电流源延时模型的接收模型负载值能够在过渡期间对电容数值进行动态调节,同时电容数值因输入斜率、输出负载和逻辑单元状态的不同,最终能精确地反映出逻辑门的输出状态。

复合电流源延时模型的信息提取方法与非线性延时模型类似:指定输入信号转换时间和逻辑门的输出端负载值,通过电路仿真模拟出逻辑门的延时和输出转换时间。但是复合电流源延时模型提取的是输入端口的电流值和输出负载的电流值,同时输入端口的电流值用于构成接收模型数据,输出负载的电流值用于构成驱动模型数据,如图 3-13 所示。

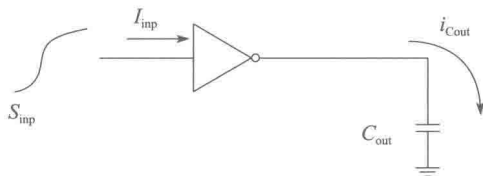


图 3-13 复合电流源延时模型仿真模型

如图 3-13 所示,根据不同的输入信号转换时间 S_{imp} 和逻辑门输出端负载值 C_{out} 的组合,在不同时间点上采样出通过输出端负载值 C_{out} 的电流值 I_{cout} 和通过输入端口的电流值 I_{imp} 。如图 3-14 所示为不同时间采样点下电流值状态图。

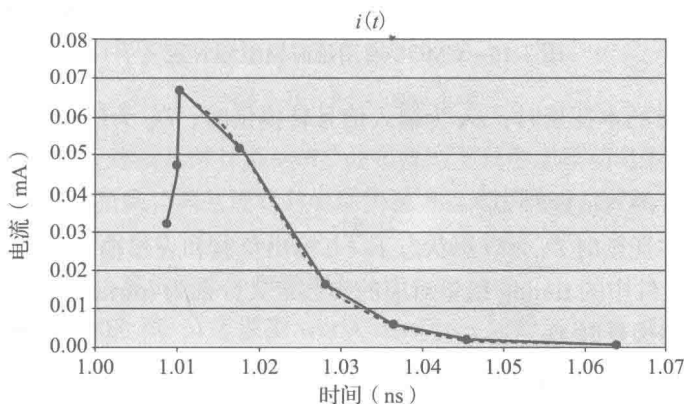


图 3-14 电流值状态图

如图 3-14 所示,复合电流源延时模型的接收模型的两个电容负载分别对应两段不同斜率的电流采样线段。复合电流源延时模型的接收模型中两个电容负载值信息的基本提取方法是:默认情况下,定义第一个电容负载值为输入端口转换波形在 0 ~ 50% 时所对应的输出端的电容负载值仿真结果,第二个电容负载值为输入端口转换波形在 50% ~ 100% 时所对应的输出端的电容负载值仿真结果。

3.3 延时计算模型

时序分析中需要基于延时模型来进行计算。根据应用和计算精度的不同,目前常用的延时计算模型主要分为以下两种。

1. CMOS 通用延时计算模型

CMOS 通用延时计算模型将时序路径的延时分成 4 个部分,总延时是这 4 部分延时之和。计算式 (3-9) 如下:

$$D_{\text{total}} = D_I + D_S + D_C + D_T \quad (3-9)$$

这 4 部分延时各自的定义如图 3-15 所示。

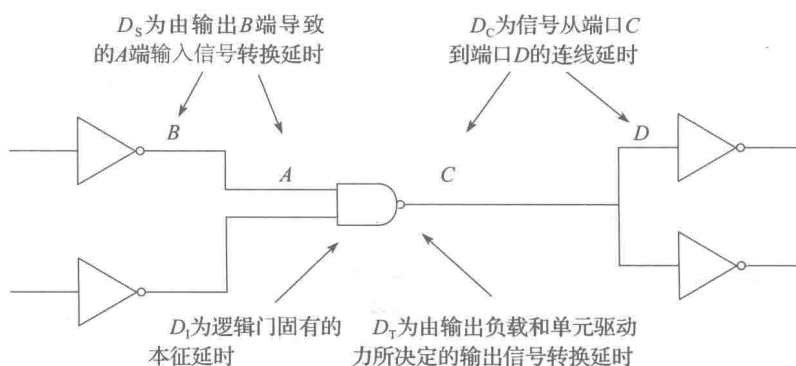


图 3-15 CMOS 通用延时模型延时定义

D_I 为逻辑门固有的本征延时; D_S 为输入信号转换延时; D_C 为信号连线延时; D_T 为输出信号转换延时。

(1) 逻辑门固有的本征延时 D_I

逻辑门固有的本征延时 D_I 为理想状态下 (0 输出负载和理想输入驱动) 逻辑门的求值延时,它在时序信息文件中的 timing 组里对应的参数定义分别为 intrinsic_rise 和 intrinsic_fall。

(2) 输入信号转换延时 D_S

输入信号转换延时 D_S 的计算公式如式 (3-10) 所示:

$$D_S = S_S \times D_{T(\text{prevstage})} \quad (3-10)$$

其中,参数 S_S 为信号转换因子。由于信号转换存在上升和下降两个状态,所以在时序信息文件中的 timing 组里对应的参数定义分别为 slope_rise 和 slope_fall。

参数 $D_{T(\text{prevstage})}$ 为前级驱动输出信号转换延时。

(3) 输出信号转换延时 D_T

输出信号转换延时 D_T 的计算公式如式 (3-11) 所示。

$$D_T = R_{\text{driver}}(C_{\text{wire}} + C_{\text{pins}})/\text{number_non_three_state_drivers} \quad (3-11)$$

其中, 参数 R_{driver} 为计算输出信号转换延时中调用的负载电阻值。同样, 由于信号转换存在上升和下降两个状态, 所以在时序信息文件中的 timing 组里对应的参数定义分别为 rise_resistance 和 fall_resistance。

参数 C_{wire} 为计算输出信号转换延时所调用的线电容负载值, 它在时序信息文件中的 wire_load 组里对应的参数定义为 capacitance 和 fanout_length。

参数 C_{pins} 为计算输出信号转换延时所调用的驱动端口电容负载值, 它在时序信息文件中的 pin 组里对应的参数定义为 capacitance。

参数 number_non_three_state_drivers 为提供输出信号的驱动源个数。

(4) 信号连线延时 D_c 。

前面介绍过互连线延时计算模型有 3 种: best_case_tree、worst_case_tree 和 balanced_tree。因此信号连线延时 D_c 基于不同的互连线延时计算模型, 分为以下 3 种计算公式。

1) 基于互连线延时计算的 RC 分布模型 best_case_tree

互连线延时计算的 RC 分布模型 best_case_tree 如图 3-16 所示。

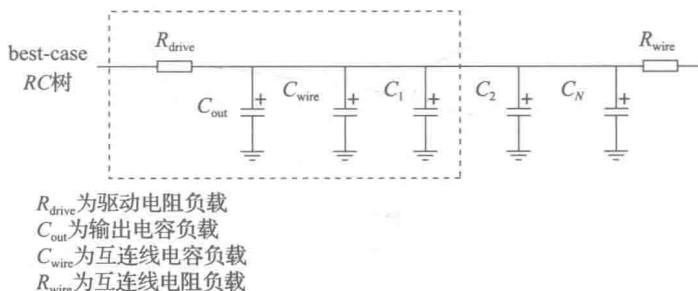


图 3-16 best_case_tree

如图 3-16 所示, 该互连线延时计算的 RC 分布模型下只考虑驱动电阻负载、互连线总电容负载和对应驱动负载电容负载, 其互连线总电阻负载 R_{wire} 的值为 0, 因此 D_c 的计算公式如下。

$$D_{c_{\text{best}}} = R_{\text{wire}} (C_{\text{wire}} + C_{\text{pin}}) = 0$$

2) 基于互连线延时计算的 RC 分布模型 worst_case_tree

基于互连线延时计算的 RC 分布模型 worst_case_tree 如图 3-17 所示。

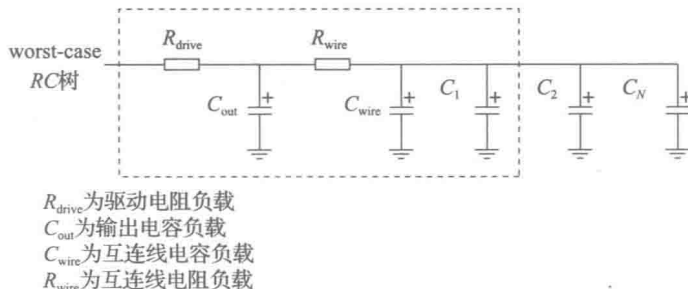


图 3-17 worst_case_tree

如图 3-17 所示, 该互连线延时计算的 RC 分布模型下, 需要考虑驱动电阻负载、互连线总电容负载、驱动负载总电容负载和互连线总电阻负载, 因此 D_C 的计算公式如式 (3-12) 所示。

$$D_{C_{\text{worst}}} = R_{\text{wire}} \left(C_{\text{wire}} + \sum_{\text{pins}} C_{\text{pin}} \right) \quad (3-12)$$

3) 基于互连线延时计算的 RC 分布模型 `balanced_tree`

基于互连线延时计算的 RC 分布模型 `balanced_tree` 如图 3-18 所示。

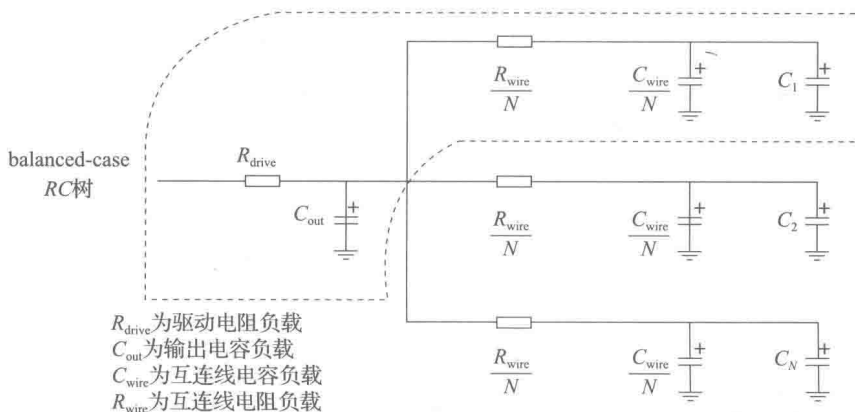


图 3-18 `balanced_tree`

如图 3-18 所示, 该互连线延时计算的 RC 分布模型下, 互连线总电阻电容负载值根据负载节点的个数多少进行了平均分配, 因此 D_C 的计算公式如式 (3-13) 所示。

$$D_{C_{\text{balanced}}} = \frac{R_{\text{wire}}}{N} \left(\frac{C_{\text{wire}}}{N} + C_{\text{pin}} \right) \quad (3-13)$$

参数 R_{wire} 为计算输出信号转换延时时所调用的互连线总电阻负载, 它在时序信息文件中的 `wire_load` 组里对应的参数定义为 `resistance` 和 `fanout_length`。

参数 C_{wire} 为计算输出信号转换延时时所调用的互连线电容负载, 它在时序信息文件中的 `wire_load` 组里对应的参数定义为 `capacitance` 和 `fanout_length`。

参数 C_{pins} 为计算输出信号转换延时时所调用的驱动端口电容负载, 它在时序信息文件中的 `pin` 组里对应的参数定义为 `capacitance`。

在实际时序分析中, 时序分析工具在计算总延迟时, 还需要考虑工艺、温度和电压变化的情况, 因此对 4 个部分需要分别乘以一个比例系数, 计算式 (3-14) 如下。

$$D_{\text{scaled}} = D \times (1 + \Delta_{\text{process}} \times K_{\text{process}}) \times (1 + \Delta_{\text{temp}} \times K_{\text{temp}}) \times (1 + \Delta_{\text{voltage}} \times K_{\text{voltage}}) \quad (3-14)$$

其中, Δ_{process} 、 Δ_{temp} 、 Δ_{voltage} 表示在通常条件下的工艺、温度与电压属性与工艺库指定的属性之间的差异, 在时序信息文件中对应的参数定义为 `nom_process`、`nom_temperature` 和 `nom_voltage`。

K_{process} 、 K_{temp} 、 K_{voltage} 表示工艺、温度和电压的比例因子。比例因子根据不同时序计算需求在时序信息文件中存在不同的比例因子定义, 部分比例因子的定义如下。

```

/* k-factors */
k_process_rise_transition
k_temp_rise_transition
k_volt_rise_transition
k_process_fall_transition
k_temp_fall_transition
k_volt_fall_transition
k_process_setup_rise
k_temp_setup_rise
k_volt_setup_rise
k_process_setup_fall

```

因此, 含有比例因子的总延时计算公式如式 (3-15) 所示。

$$D_{\text{total(scaled)}} = D_I(\text{scaled}) + D_S(\text{scaled}) + D_C(\text{scaled}) + D_T(\text{scaled}) \quad (3-15)$$

CMOS 通用延时计算模型是一个线性延时模型, 但是在真实电路中, 逻辑门延时与输入信号跳变时间以及输出负载之间并不是遵循严格的线性关系, 因此该延时计算模型不适合于对精度要求较高的延时计算。

2. CMOS 非线性延时计算模型

CMOS 非线性延时计算模型 (NLDM) 是一种时序计算精度较高的延时计算模型。因此, 在 Synopsys 工艺库模型中, 目前业界主要使用非线性延时计算模型。该延时模型由输入信号转换时间与输出负载作为索引, 时序分析时以一个二维查找表的形式来计算延时, 查找表中时序数据的实际构成如图 3-19 所示。

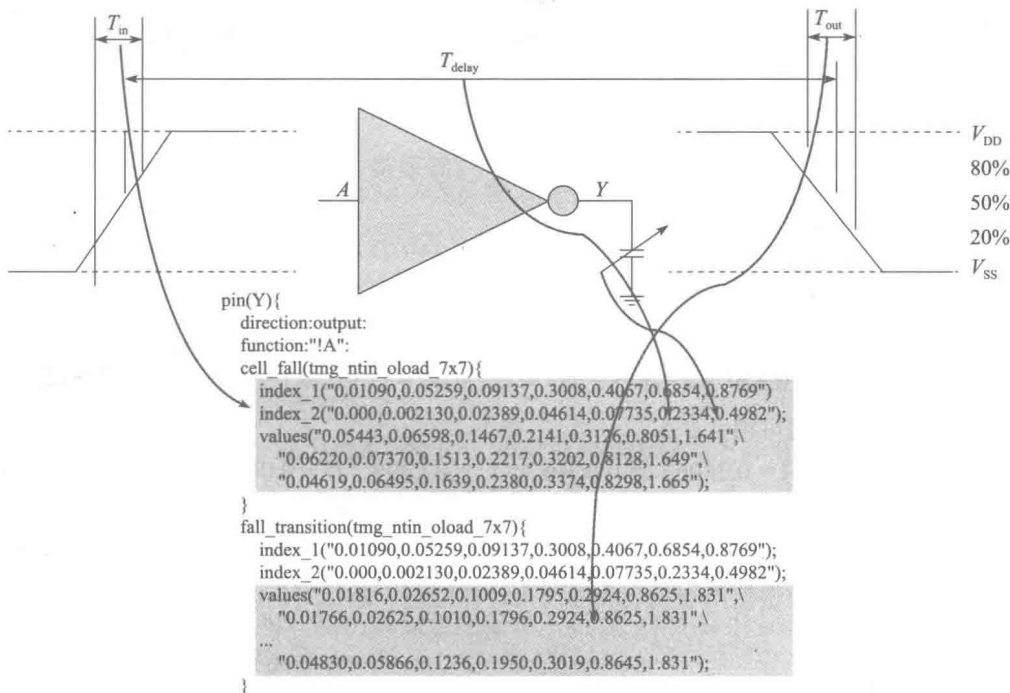


图 3-19 查找表构成

基于非线性延时计算模型通常的计算过程为：查找表中索引参数的某个采样点延时可直接查表得出，再通过插值算法来计算延时。通过非线性延时计算模型计算所能达到的精度有赖于采样点选取的合理性以及所用的插值算法。只要采样点较为合理，采用一般的多项式插值算法就能取得较好的延时结果。延时的一般插值计算方法如图 3-20 所示。

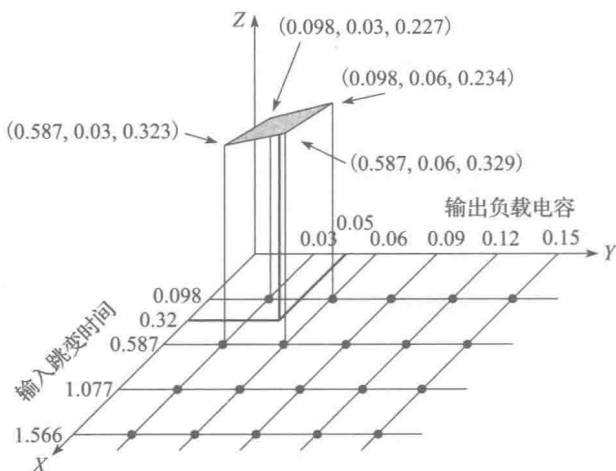


图 3-20 插值计算方法

三维坐标中， X 为输入端口信号的跳变时间 Transition time (ns)， Y 为输出端负载电容值 C_{Load} (pf)， Z 为延时值 T_{delay} (ns)，计算式 (3-16) 如下。

$$Z = A + B \cdot X + C \cdot Y + D \cdot X \cdot Y \quad (3-16)$$

其中，系数 A 、 B 、 C 、 D 可通过 Gauss 消元法得到。

如果要计算 $X = 0.32$ 、 $Y = 0.05$ 时的 Z 值，只需取最接近 X 、 Y 的 4 个采样点，按照公式 3-16 构建如下线性方程组：

$$\begin{cases} 0.227 = A + B \cdot 0.098 + C \cdot 0.03 + D \cdot 0.098 \cdot 0.03 \\ 0.234 = A + B \cdot 0.098 + C \cdot 0.06 + D \cdot 0.098 \cdot 0.06 \\ 0.323 = A + B \cdot 0.587 + C \cdot 0.03 + D \cdot 0.587 \cdot 0.03 \\ 0.329 = A + B \cdot 0.587 + C \cdot 0.06 + D \cdot 0.587 \cdot 0.06 \end{cases}$$

通过解线性方程组可得：

$$A = 0.2006, B = 0.1983, C = 0.2399, D = 0.0677$$

再将 A 、 B 、 C 、 D 以及 X 、 Y 的值代入公式 3-16，可得该点的延时值 Z 如下：

$$Z = 0.2006 + 0.1983 \times 0.32 + 0.2399 \times 0.05 + 0.0677 \times 0.32 \times 0.05 = 0.275$$

非线性延时计算模型使用查找表和插值来计算延迟。总延时由两部分组成，如式 (3-17) 所示。

$$D_{total} = D_{cell} + D_C \quad (3-17)$$

这两部分延时各自的相关定义如图 3-21 所示。

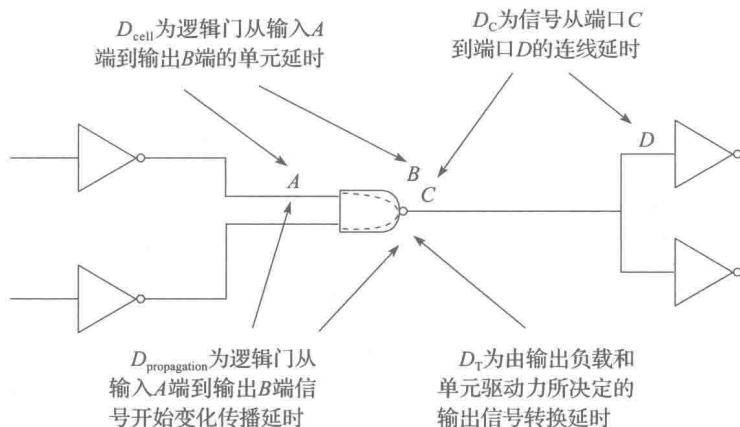


图 3-21 非线性延时模型的延时定义

由图 3-21 所示, D_{cell} 为逻辑门延时, D_C 为信号互连线延时, $D_{\text{propagation}}$ 为逻辑门传播延时, $D_{\text{transition}}$ 为逻辑门转换延时。

□ 信号互连线延时 D_C

信号互连线延时 D_C 可以通过互连线延时计算模型来计算, 前面已经介绍过。

□ 逻辑门转换延时 $D_{\text{transition}}$

逻辑门转换延时 $D_{\text{transition}}$ 可以根据实际设计需求, 一般定义为信号转换时从高电平的 20% 到 80% 切换中的延时, 也可以定义从高电平 10% 到 90% 或者从 30% 到 70% 的延时, 该百分比值可以根据设计需求进行定义。它在时序信息文件中的对应的查找表名称分别为 `rise_transition` 和 `fall_transition`。

□ 逻辑门传播延时 $D_{\text{propagation}}$

逻辑门传播延时 $D_{\text{propagation}}$ 一般定义为从输入信号的 50% 到输出转换至高电平的 10% (上升沿) 或 90% (下降沿) 的延时, 该百分比值可以根据设计需求进行定义。它在时序信息文件中的对应的查找表名称分别为 `rise_propagation` 和 `fall_propagation`。

□ 逻辑门延时 D_{cell}

逻辑门延时 D_{cell} 一般定义为输入和输出波形 50% 翻转点之间的时间, 如图 3-22 所示, 该百分比值可以根据设计需求进行定义。

如图 3-22 所示, T_{delay} 即为逻辑门延时, 也是信号在上升与下降电平的 50% 之间的延时。它在时序信息文件中对应的查找表名称分别为 `cell_rise` 和 `cell_fall`。

非线性延时计算模型提供两种计算 D_{cell} 方法, 一个工艺库中可以同时包含这两种方法所使用的信息。

1) 将逻辑门转换时间和输出端负载电容值 (C_{load}) 作为索引, 通过查找表得到相应数据, 并通过插值计算直接得到延时。

用非线性延时计算模型计算 D_{cell} 的例子如图 3-23 所示。

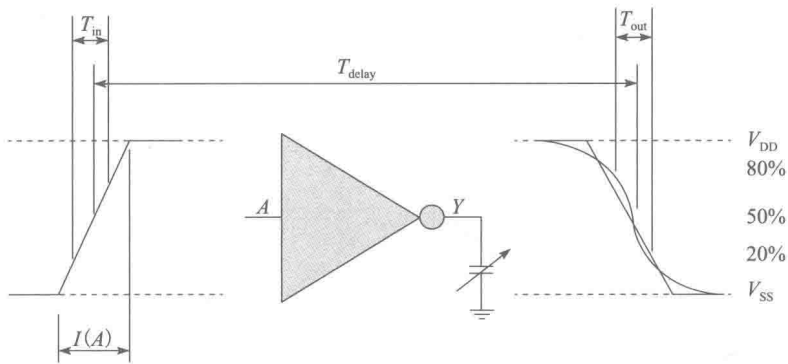


图 3-22 逻辑门延时

如图 3-23 所示，计算逻辑门 U_1 的 D_{cell} 值时，先确定输入端口的转换时间， U_1 输入端口的转换时间由前级单元 U_0 的输出决定。当 U_0 的输出为上升（rise）转换时间时，由于 U_1 的时序弧为反向时序弧，因此 U_1 的逻辑门延时应该在对应的 U_1 下降时序弧查找表中进行插值计算。由于下降时序弧查找表是二维表，因此还需要确定 U_1 输出端负载值，通过互连线负载模型和驱动节点负载值得到输出端总负载值。最后，根据 U_1 输入端口转换时间和输出端负载电容值作为索引，在对应的 U_1 下降时序弧二维查找表得到相应数据，并进行插值计算直接得到延时。

2) 将逻辑门传播时间（propagation time）和输出端负载电容值（ C_{load} ）作为索引，通过查找表得到相应数据，并进行插值计算先得到逻辑门传播延时和逻辑门转换延时，再按式（3-18）计算出逻辑门延时 D_{cell} 。

$$D_{cell} = D_{propagation} + D_{transition} \tag{3-18}$$

逻辑门传播延时 $D_{propagation}$ 和逻辑门转换延时 $D_{transition}$ 在前面已经介绍。

如图 3-24 所示， D_p 为从输入信号的 50% 到输出转换至高电平的 10%（上升沿）的逻辑门传播延时。 D_t 为输出信号在高电平的 20% 与 50% 转换中的逻辑门转换延时。 D_p 和 D_t 都是通过逻辑门转换时间和输出端负载电容值索引在查找表中得到相应数据，再进行插值计算得到的。

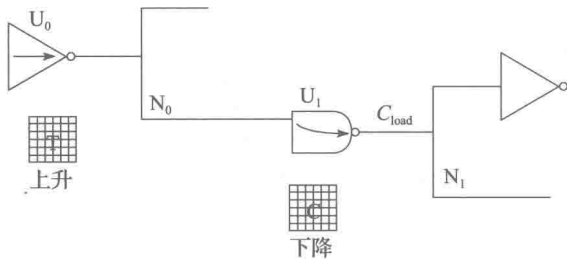


图 3-23 非线性延时计算模型计算

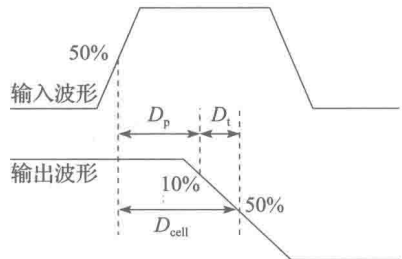


图 3-24 逻辑门延时

同样，在基于非线性延时计算模型的时序分析中，实际时序分析工具在计算总延迟时，

还需要考虑工艺、温度和电压变化的情况,因此,4个部分需要分别乘以一个比例系数,计算式(3-19)如下。

$$D_{\text{scaled}} = D \times (1 + \Delta_{\text{process}} \times K_{\text{process}}) \times (1 + \Delta_{\text{temp}} \times K_{\text{temp}}) \times (1 + \Delta_{\text{voltage}} \times K_{\text{voltage}}) \quad (3-19)$$

其中 Δ_{process} 、 Δ_{temp} 、 Δ_{voltage} 表示在通常条件下的工艺、温度与电压属性与工艺库指定的属性之间的差异,在时序信息文件中对应的参数定义为 nom_process 、 nom_temperature 和 nom_voltage 。

K_{process} 、 K_{temp} 、 K_{voltage} 表示工艺、温度和电压的比例因子。比例因子根据有时序计算需求在时序信息文件中有不同的比例因子定义。

3.4 互连线计算模型

当逻辑门完成输入信号的求值转换并输出后,输出信号通过信号线传播到下一级逻辑门输入端的时间即为信号互连线传播延时。一组反相器对之间的信号互连线延时如图 3-25 所示。

芯片内的互连线大致可以分为以下 3 种:

1) 短线,即局部互连线。短线主要用于逻辑门之间或者用于速度不是很高的器件间的连接。通常短线的长度远远小于信号波长,短线的延时主要受寄生电容和耦合电容的影响,对系统延迟没有显著影响,一般可以忽略。

2) 中长线,即模块间互连线。中长线信号传输的速度要求比短线高,电感耦合效应也变得突出,因而容易引起很高的噪声。中长线需要采用低电阻率金属和中等厚度的绝缘介质。

3) 长线,即全局互连线。长线对电路性能起着关键作用。长线特别需要采用低电阻率金属以减小信号线和电源线的电阻损耗,需要厚的绝缘层来增加特征阻抗、减小延时,需要较宽的线间距以减少串扰。虽然宽线、宽间距可以减小 RC 延时和串扰,但也会影响布线密度。

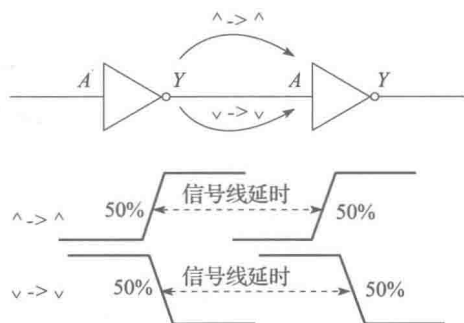


图 3-25 反相器对之间的信号线延时

3.4.1 互连线计算模型

互连线的几何尺寸和拓扑结构会对互连线的电气特性——电容、电阻、电感——产生影响,从而影响互连线延时、功耗和可靠性。因此,需要建立用于分析互连线的实际特性和参数间关系的电气模型。目前,常见的互连线模型有以下几种。

1. 集总 C 模型

当导线的电阻部分很小并且驱动单元的转换频率在较低范围内时,就可以很合理地只考虑导线的电容部分,并把分布的电容集总为单个电容。集总 C 模型如图 3-26 所示。

集总 C 模型计算最简单,但在超深亚微米的设计工艺下精确度不高。

2. 集总 RC 模型

当导线长度超过几微米后就会具有比较明显的电阻，因此必须采用电阻 - 电容模型。集总 RC 模型如图 3-27 所示。

集总 RC 模型的计算也同样简单，在超深亚微米的设计工艺下同样精确度不高，Elmore 延时偏离 SPICE 模拟结果 33% 以上。

集总 RC 模型中，大多数电路都可以标识为一个 RC 树，树状结构的 RC 网络如图 3-28 所示。

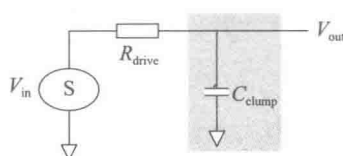


图 3-26 集总 C 模型

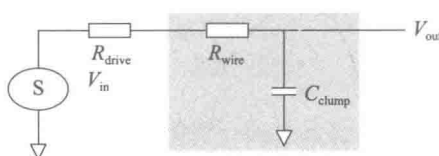


图 3-27 集总 RC 模型

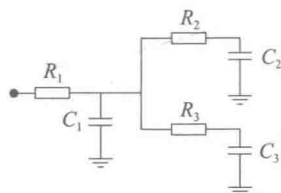


图 3-28 简单的 RC 树

其 Elmore 延时表达式如式 (3-20) 所示。

$$T_{Di} = \sum_k C_k R_{ik} \quad (3-20)$$

其中， C_k 表示网络结构中所有的电容， R_{ik} 是从节点 i 到节点 k 的路径中所共有的电阻。如图 3-28 所示节点 2 处的 Elmore 延时如式 (3-21) 所示。

$$T_{D2} = C_1 R_1 + C_2 (R_1 + R_2) + C_3 R_1 \quad (3-21)$$

Elmore 延时是基于概率的理论来求解互连线的延迟的，其主要特点如下：

- 1) 非常简单，其所有的运算仅仅为加法和乘法。
- 2) 精度不高，Elmore 延时是互连线 50% 延时的上界，即 Elmore 延时是一种比较悲观的互连延迟模型。
- 3) Elmore 不能处理电感，不能直接处理悬浮电容以及耦合电感。
- 4) 在 RC 树中，距离驱动节点越远的节点估算的精度越高。
- 5) Elmore 延时的计算精度会受输入信号的影响，输入信号的转换时间越长，估算的精度越高。

由于 Elmore 延时模型的简洁性，并且在大部分低要求设计中具有较高的精度，因此在初期电路设计和验证中被广泛应用。

3. 分布 RC 模型

分布 RC 模型是时序分析中比较常用的互连线模型，如图 3-29 所示。



图 3-29 分布 RC 模型

分布 RC 模型主要分为 L 模型、 π 模型和 T 模型 3 种。其中 L 模型如图 3-29 所示，其计算公式如式 (3-22) 所示。

$$\begin{aligned} R_l &= R_p * L \\ C_l &= C_p * L \end{aligned} \quad (3-22)$$

其中， R_p 为单位长度的电阻值， C_p 为单位长度电容值， L 为互连线长度。L 模型是一种较差的分布 RC 模型，因为它需要计算大量的小段互连线才能得到精确的结果。

π 模型是把电容值为 C_l 的总电容分成均匀的两个电容值，即为 $C_l/2$ 的子电容，并且子电容位于电阻 R_l 的两端，如图 3-30 所示。

相比 L 模型， π 模型在计算精确度上要好得多，把连线分成 3 段就足以得出与 SPICE 模拟结果相对比后精确到 3% 的结果。

T 模型是把总电阻 R_l 分成均匀的两个电阻值为 $R_l/2$ 的子电阻，并且子电阻位于电容 C_l 的两端，如图 3-31 所示。

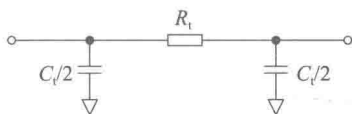


图 3-30 π 模型

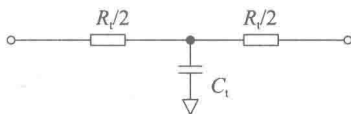


图 3-31 T 模型

T 模型与 π 模型在精确度上结果相当，但是 π 模型所形成的电路将多一个节点。因此在同样的导线分段数量下，T 模型延时计算求解速度比 π 模型要快。

4. 传输线模型

在深亚微米工艺下，为保持信号的完整性，传输线模型将逐渐占据主导。加入电感因素后，在计算精度得到提高的同时，也增加了计算量，如图 3-32 所示。

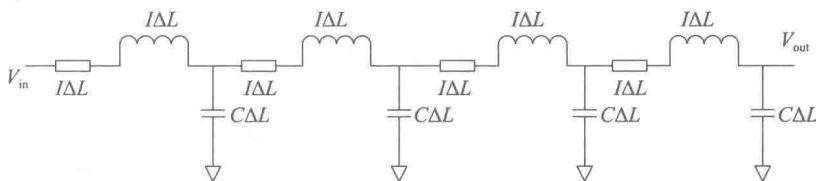


图 3-32 传输线模型

传输线模型相对比较精确，但计算量太大。同时由于忽略与邻近对象的耦合，简化的传输线模型的精确度有所下降。当然，传输线模型可以用足够数量的 RLC 分段逼近，RLC 分段数越多，则越接近传输线模型行为。

3.4.2 线负载时序模型

在 RTL 代码综合阶段进行静态时序分析时，由于互连线还没有物理信息，因此可以通过调用线负载模型来估算实际物理实现后的线负载大小，用于静态时序分析时计算互连线延

时。线负载模型在时序信息文件中的内容如下。

```
wire_load("...") {
    resistance : ..;
    capacitance : ...;
    area : ...;
    slope : ...;
    fanout_length (...);
}
```

其中参数 `wire_load` 定义线负载模型的名称, 参数 `resistance` 定义互连线单位电阻值, 参数 `capacitance` 定义互连线单位电容值, 参数 `area` 定义互连线单位长度的面积值, 参数 `slope` 定义扩展斜率值, 参数 `fanout_length` 定义对应扇出大小的互连线长度。

基于线负载模型计算互连线负载大小的过程如下:

1) 定义线负载模型 `wl30`, 如下所示。

```
wire_load ("wl30") {
    resistance : 5.0;
    capacitance : 1.1;
    area : 0.05;
    slope : 0.5;
    fanout_length (1, 2.6);
    fanout_length (2, 2.9);
    fanout_length (3, 3.2);
    fanout_length (4, 3.6);
    fanout_length (5, 4.1);
}
```

2) 一条扇出数目为 8 的互连线的负载大小计算方法如图 3-33 所示。

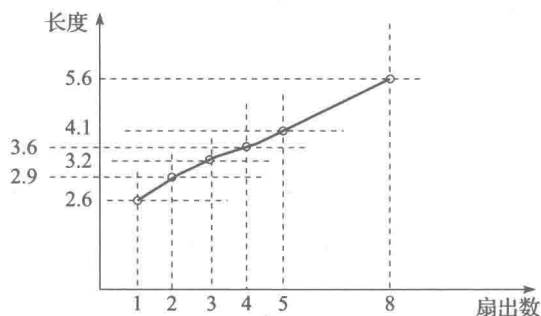


图 3-33 扇出数目为 8 的互连线计算

由图 3-33 可以计算得到如下信息:

$$\begin{aligned} \text{互连线长度} &= \text{扇出数目为 5 的互连线长度} + (\text{扇出 8} - \text{扇出 5}) \times \text{扩展斜率值} \\ &= 4.1 + (8 - 5) \times 0.5 = 5.6 \end{aligned}$$

$$\text{互连线电容} = \text{互连线长度} \times \text{互连线单位电容值} = 5.6 \times 1.1 = 6.16$$

互连线电阻 = 互连线长度 \times 互连线单位电阻值 = $5.6 \times 5.0 = 28.0$

互连线面积 = 互连线长度 \times 互连线单位长度面积值 = $5.6 \times 0.05 = 0.28$

3.5 引脚电容值的计算

每一个单元的时序模型中除了包含延迟信息外, 还包含输入引脚的电容信息。

在输入上升和下降中, 不同转换时间和不同负载情况下的电容值都是不一样的。引脚电容参数的定义如图 3-34 所示。

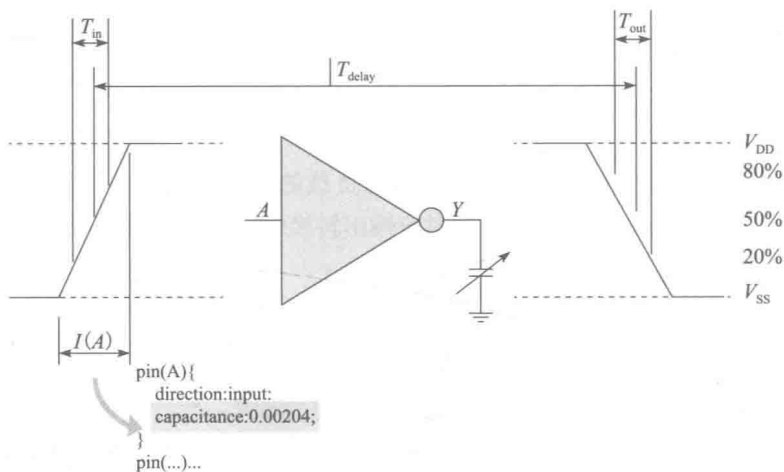


图 3-34 引脚电容参数

引脚电容值的计算方法一般分为以下两种:

1) 引脚电容是流入输入引脚电流的积分, 如图 3-35 所示。

所以计算式 (3-23) 如下。

$$C = (I_{cap} - I_{Leakage}) \frac{\Delta T}{\Delta V} \Big|_{\substack{\text{cin_high_threshold} * V_{DD} \\ \text{cin_low_threshold} * V_{DD}}} \quad (3-23)$$

其中, 积分区间为输出信号电平转换的区间, 如图 3-36 所示。

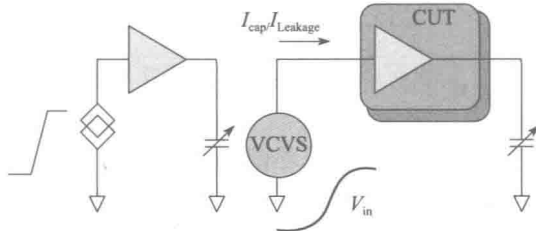


图 3-35 PIN 脚电容仿真模型

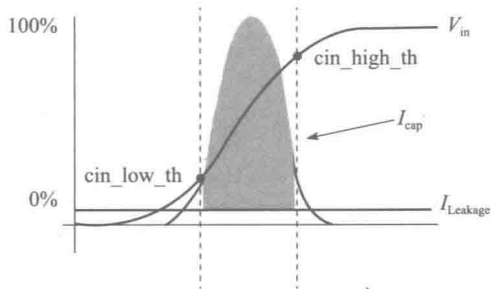


图 3-36 输出信号电平转换区间

2) 基于输出转换时间结果, 通过查找表的方法来确定所对应的输出电容负载, 如图 3-37 所示。

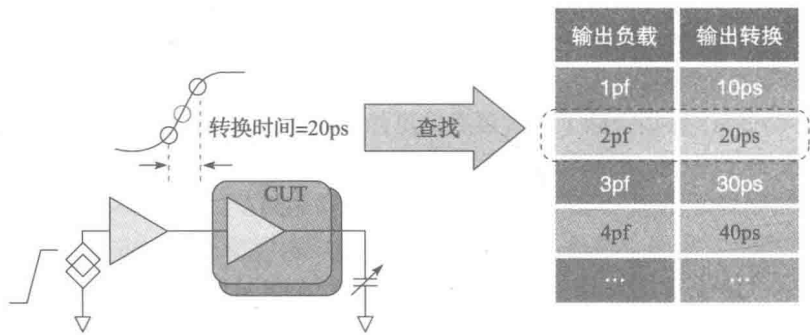


图 3-37 查找表方法

通过电路仿真建立输出转换时间与输出电容负载的对应查找表, 如图 3-37 所示。当输出转换时间为 20ps 时, 可在查找表中找到对应输出转换时间为 20ps 的输出电容负载为 2pf。

3.6 功耗模型的计算

功耗计算包括开关功耗 I_{sw} , 短路电流功耗 I_{sc} , 亚阈漏流功耗 I_{lk} , 如图 3-38 所示。

动态功耗由开关功耗与短路功耗组成, 动态功耗与输入输出引脚的时序弧有关, 同时通过查找表以输出负载与输入信号转换时间为索引联系起来。

动态功耗是单元在转换过程中电源电流的积分, 如图 3-39 所示。

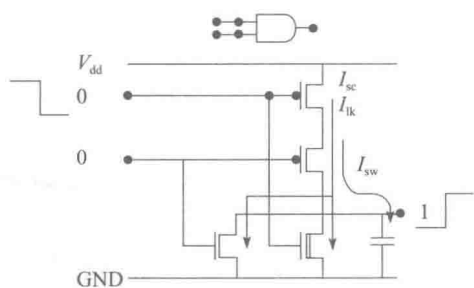


图 3-38 功耗计算参数

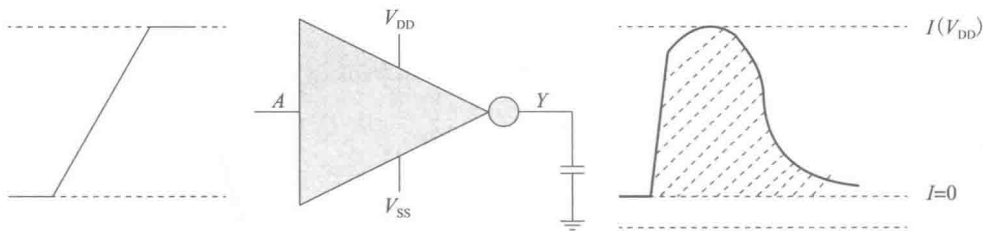


图 3-39 动态功耗

静态功耗为亚阈漏流功耗, 静态功耗与单元类型及单元所处的逻辑状态有关。
亚阈漏流是逻辑单元在稳定状态下消耗功耗的原因, 亚阈漏流功耗的计算方法是: 设置单元的输入信号为固定电平, 然后对单元的亚阈漏流进行瞬态分析, 来得到亚阈漏流功耗

值,如图 3-40 所示。

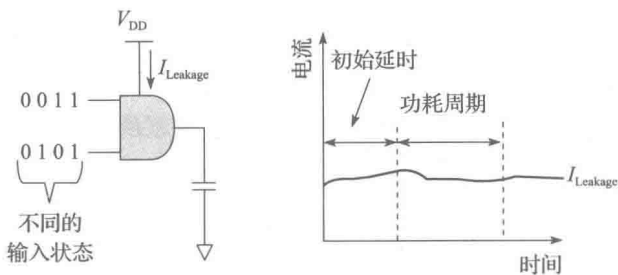


图 3-40 哑阈漏流

功耗信息查找表实际的结构形式如图 3-41 所示。

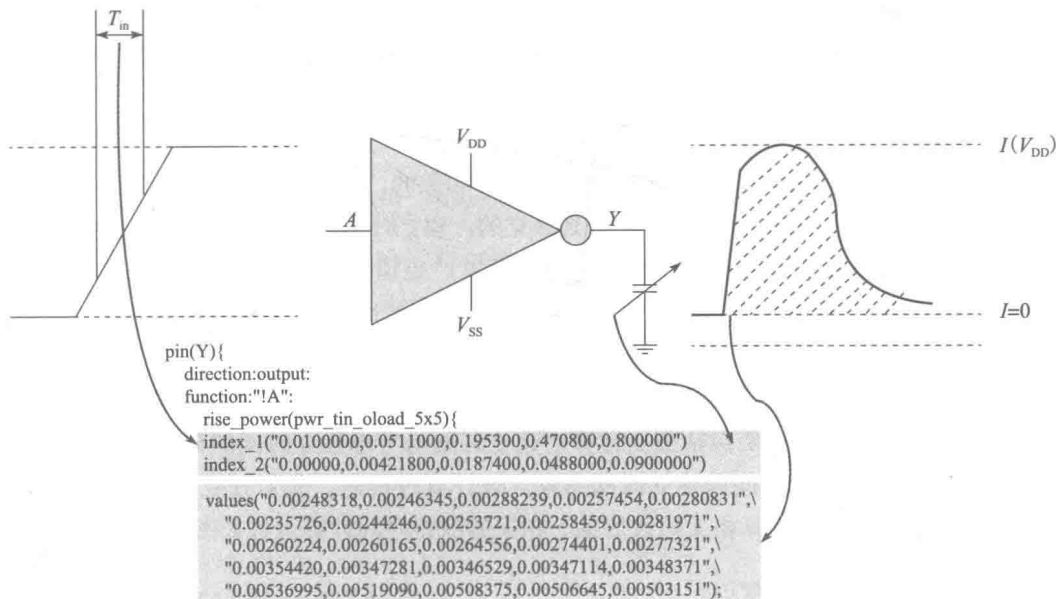


图 3-41 功耗信息查找表

3.7 时序信息建模基本方法

时序模型需要考虑两方面的延迟信息:一个是输入端口到输出端口的延迟信息,即时序弧;一个是输入端口之间存在的时序约束信息,比如建立或保持时间约束,即约束时序弧。

时序信息建模过程中,当一个模块规模较大且输入输出端口数较多时,通常需要对其时序弧进行化简,减少延迟弧和约束弧的数量,减少时序建模时工具计算所需的时间以及存储空间。为此,时序信息建模定义了两种基本化简操作。

1) 时序弧串联合并：对于两条首尾相接的时序弧，如图 3-42 所示。

延迟值分别为 d_1 和 d_2 ，可以将这两条时序弧合并产生一条新的时序弧，起点为第一条时序弧的起点，终点为第二条时序弧的终点，延迟值为两条时序弧的延迟之和 ($d_1 + d_2$)。同理，对于时序弧条数多于两条串联的情况，可将各时序弧的延迟之和依次相加作为新时序弧的值。

2) 时序弧并联合并：对于连接两个端点的两条以上并联的时序弧，如图 3-43 所示。



图 3-42 时序弧串联合并



图 3-43 时序弧并联合并

可以将这两条时序弧合并产生一条新的时序弧，延迟值为两条时序弧上延迟较大的一个。同理，对于两条以上时序弧并联的情况，选择其中最大的延迟作为新时序弧的值。

建立时序模型就是要表示出每一条延迟弧和约束弧对应的值，该值可以是线性函数或查找表。

1. 基于独立的时序提取方法

独立的时序约束指的是每个约束信息之间都是不相关的，这样可以通过常规直接的方法进行计算，比如建立时间与保持时间的计算都是独立的。建立时间计算的关系图如图 3-44 所示。

建立时间的最终确定基于输入信号在时钟最近处产生错误的时间点上，建立时间的计算过程中不考虑保持时间的影响，这种独立的时序约束会导致比较偏激的结果。

2. 基于独立与相关混合的时序信息提取方法

独立与相关混合的约束指的是：有的约束不考虑其他约束的影响，而有些约束需要考虑其他约束的影响，其类型主要包括独立计算的建立时间与考虑建立时间的保持时间约束，独立的保持时间与考虑保持时间的建立时间约束。该计算过程是，先对独立的约束进行计算，再基于该独立的约束信息计算相关约束信息。相关的保持时间计算过程如图 3-45 所示。

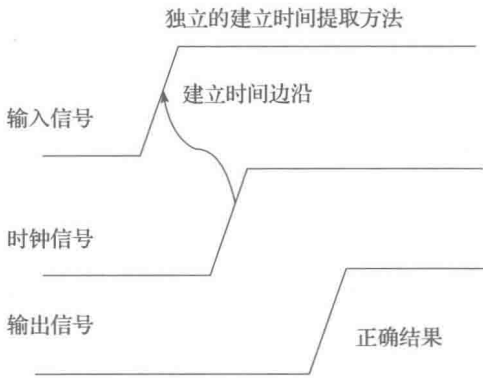


图 3-44 独立的建立时间计算

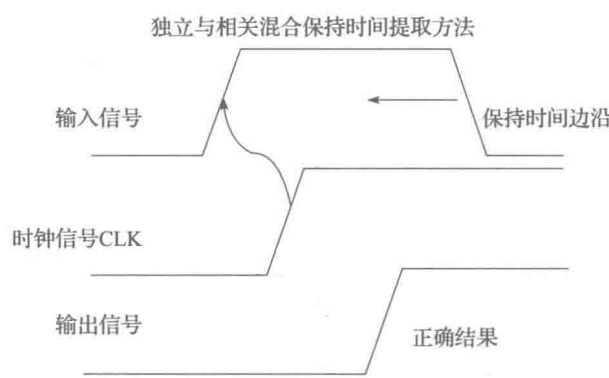


图 3-45 相关的保持时间计算

首先对独立的建立时间约束进行计算，再通过基于建立时间的信息对保持时间约束进行计算，该方法能够得出比较保守的结果。

3. 基于完全相关的时序信息提取方法

完全相关的时序约束指在对建立时间与保持时间进行计算时，同时考虑它们之间的相关性。该计算首先进行初始化参数设置，使结果同时满足保持与建立时间约束的要求。相关的保持与建立时间的计算过程如图 3-46 所示。

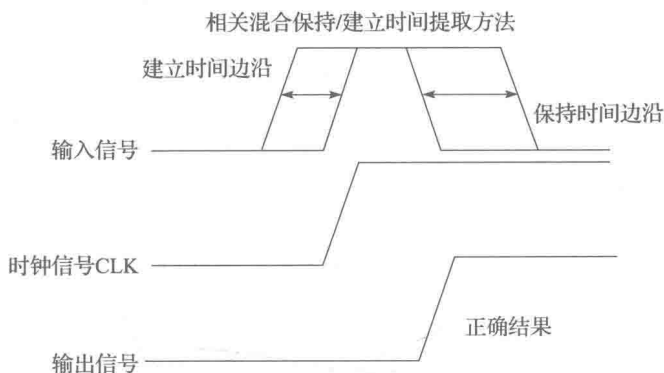


图 3-46 相关的保持与建立时间的计算

首先通过一个很小的延时脉冲进行初始化，延时脉冲的宽度保证所有约束都是符合的。在建立时间的边缘，减少保持时间的宽度来找到符合两个约束的结果，该结果比独立约束要保守。

第 4 章

时序信息库文件

时序信息库 (timing library) 文件中记录着逻辑门延时、输出信号转换延时和功耗等信息, 这些信息在用于时序分析时被调用, 来计算延时值和功耗值。时序文件的内容主要由组 (group)、属性 (attribute) 和因子 (factor) 组成。

前面介绍过, 根据应用和计算精度的不同, 目前常用的 Synopsys 工艺库模型主要分为以下 3 种: 线性延时模型 (linear delay model)、非线性延时模型 (non-linear delay model) 和复合电流源延时模型 (composite current source delay model)。对于不同的 Synopsys 工艺库模型存在不同的时序文件内容。由于线性延时模型并不是主流使用的 Synopsys 工艺库模型, 因此本章只介绍非线性延时模型和复合电流源延时模型时序文件内容定义。

4.1 非线性延时模型

由于组、属性和因子的种类和数量很多, 而且不是全都需要同时在时序文件中定义, 下面只介绍比较重要的组、属性和因子。

关于非线性延时模型文件的详细内容, 可以参考附录中 NLDM 时序信息文件片段内容。

4.1.1 库组

库组 (library group) 是时序信息文件的最顶层定义, 库组的定义处在时序信息文件的第一行, 所有的时序文件内容都在库组里面定义。根据设计的需求, 可以为库组定义对应的库组名称, 用于区分不同的时序信息库文件, 如下所示。

```
library (my_library) {  
.....  
}
```

在库组的内容里存在基本的属性定义, 比如工艺属性 (technology attribute)、延时模型属性 (delay_model attribute)、延时和转换斜率属性 (Delay and Slew Attributes) 等。这里只对几个主要的属性进行说明。

1. 工艺属性

工艺属性定义用来区别时序分析的设计是基于那种类型的设计。目前有两种类型的设计需要时序分析，一种是 ASIC 设计，一种是 FPGA 设计。但是它们在设计流程上有很大的不同，所以在进行时序分析时也有所区别。因此工艺属性有两个设置值可以选择：CMOS 和 FPGA。该属性定义的示例如下。

```
technology (cmos);
```

如果库组中没有定义工艺属性，那么默认设置是 CMOS。

2. 延时模型属性

时序分析中需要基于延时模型来进行计算，因此需要在库组中定义延时模型。目前时序分析延时模型分为 CMOS 通用延时计算模型和 CMOS 非线性延时计算模型 (NLDM) 两种，因此延时模型属性有两个设置值可以选择：generic_cmos 和 table_lookup。该属性定义的示例如下。

```
delay_model : table_lookup;
```

如果库组中没有定义延时模型属性，那么默认设置是 generic_cmos。

3. 延时和转换斜率属性

前面介绍过，组合逻辑门单元相关的时序参数主要包括信号转换延时和逻辑门延时，因此通过延时和转换斜率属性来进行定义。延时和转换斜率属性有 9 个参数，其中有 4 个延时属性参数：input_threshold_pct_fall、input_threshold_pct_rise、output_threshold_pct_fall 和 output_threshold_pct_rise，4 个转换斜率属性参数：slew_lower_threshold_pct_fall、slew_upper_threshold_pct_fall、slew_lower_threshold_pct_rise 和 slew_upper_threshold_pct_rise，1 个减免属性参数：slew_derate_from_library。除了减免属性参数，其他 8 个属性在前面都已经介绍。减免属性参数的设置主要用于使时序计算结果更加接近实际情况。该属性定义的示例如下。

```
slew_lower_threshold_pct_fall : 30.0;
slew_upper_threshold_pct_fall : 70.0;
slew_lower_threshold_pct_rise : 30.0;
slew_upper_threshold_pct_rise : 70.0;
input_threshold_pct_fall      : 50.0;
input_threshold_pct_rise      : 50.0;
output_threshold_pct_fall     : 50.0;
output_threshold_pct_rise     : 50.0;
slew_derate_from_library      : 0.5;
```

(1) 日期属性 (date attribute)

日期属性定义时序文件的生成日期，这样可以用来区别不同时间段的时序信息文件。该属性定义的示例如下。

```
date : "Tue Mar 9 16:58:12 2016";
```

(2) 修订版本属性 (revision attribute)

修订版本属性定义时序文件修订版本，这样可以用来区别不同版本的时序信息文件。该属性定义的示例如下。

```
revision : 1.1;
```

(3) 注释属性 (comment attribute)

注释属性用于对当前的时序信息文件中的一些重要信息进行必要的声明。该属性定义的示例如下。

```
comment : "Copyright (c) 2016 www.icdream.com, Inc. All Rights Reserved.";
```

(4) 时间单位属性 (time_unit attribute)

时间单位属性定义时序信息文件中的时间数值单位。时序信息文件中的时间数值都不带单位，其默认数值单位即为该时间单位属性所定义。该属性定义的示例如下。

```
time_unit : "1ns";
```

(5) 电压单位属性 (voltage_unit attribute)

电压单位属性定义时序信息文件中的电压数值单位。时序信息文件中的电压数值都不带单位，其默认数值单位即为该电压单位属性所定义。该属性定义的示例如下。

```
voltage_unit : "1V";
```

(6) 上拉下拉电阻单位属性 (pulling_resistance_unit attribute)

上拉下拉电阻单位属性定义时序信息文件中的上拉下拉器件的电阻数值单位。时序信息文件中的上拉下拉器件的电阻数值都不带单位，其默认数值单位即为该上拉下拉电阻单位属性所定义。该属性定义的示例如下。

```
pulling_resistance_unit : "1kohm";
```

(7) 电容负载单位属性 (capacitive_load_unit attribute)

电容负载单位属性定义时序信息文件中的电容负载数值单位。时序信息文件中的电容负载数值都不带单位，其默认数值单位即为该电容负载单位属性所定义。时序信息文件中需要用到该电容负载单位的设置包括默认电容负载设置 (default capacitances)、最大扇出电容负载设置 (max_fanout capacitances)、PIN 脚电容负载设置 (pin capacitances) 和互连线负载设置 (wire capacitances)。该属性定义的示例如下。

```
capacitive_load_unit (1.0,pf);
```

(8) 亚阈漏流功耗单位属性 (leakage_power_unit attribute)

亚阈漏流功耗单位属性定义时序信息文件中的功耗数值单位。时序信息文件中的功耗数值都不带单位，其默认数值单位即为该亚阈漏流功耗单位属性所定义。该属性定义的示例如下。

```
leakage_power_unit : "1pW";
```

(9) 单元优化替换属性 (in_place_swap_mode attribute)

设计工具需要对设计进行时序和功耗优化，最基本的方法就是把不合适的单元替换成合适的单元。但是是否可以允许设计工具进行单元替换优化操作，需要在时序信息文件中通过单元优化替换属性进行定义。因此单元优化替换属性有两个设置值可以选择：match_footprint 和 no_swapping。该属性定义的示例如下。

```
in_place_swap_mode : match_footprint;
```

优化替换的基础需要满足两个前提条件：第一个条件是替换和被替换的单元都具有相同的功能，第二个条件是替换和被替换的单元都具有相同数量和名字的输入输出端口。

(10) 工艺库特征属性 (library_features attribute)

工艺库特征属性定义其他 Synopsys 工具可以使用具有该工艺库特征定义的命令进行设计。工艺库特征属性有 5 个设置值可以选择：report_delay_calculation、report_power_calculation、report_noise_calculation、report_user_data 和 allow_update_attribute。该属性定义的示例如下。

```
library_features(report_delay_calculation);
```

工艺库特征属性的默认设置是无设置，即留空。

(11) CMOS 默认属性 (CMOS default attribute)

在时序文件中可以预先对部分属性设置为默认值。这样对于设置了默认值的属性，如果没有对应的设置进行定义，则可以根据默认属性的定义来进行调用。一般需要定义默认值的属性包括工艺、功耗、输入输出端口负载、逻辑门单元本征延时、最大扇出、最大转换时间等。该属性定义的示例如下。

```
default_operating_conditions : typical;
default_leakage_power_density attribute: 0.0;
default_cell_leakage_power attribute: 0.0;
default_fanout_load attribute: 1.0;
default_output_pin_cap attribute: 0.0;
default_inout_pin_cap attribute: 0.00158;
default_input_pin_cap attribute: 0.00158;
default_max_transition attribute: 0.744;
```

4.1.2 因子

库组中定义工艺、温度和电压的比例因子称为 K 因子，定义该比例因子的目的是使时序分析结果更加接近实际情况。

例如，在实际时序分析中，时序分析工具在计算总延迟时，还需要考虑工艺、温度和电压变化的情况，因此对 4 个部分需要分别乘以一个比例系数，计算式 (4-1) 如下。

$$D_{\text{scaled}} = D \times (1 + \Delta_{\text{process}} \times K_{\text{process}}) \times (1 + \Delta_{\text{temp}} \times K_{\text{temp}}) \times (1 + \Delta_{\text{voltage}} \times K_{\text{voltage}}) \quad (4-1)$$

其中, K_{process} 、 K_{temp} 、 K_{voltage} 即为工艺、温度和电压的比例因子, 该属性定义的示例如下。

```

k_process_cell_leakage_power : 0;
k_temp_cell_leakage_power : 0;
k_volt_cell_leakage_power : 0;
k_process_internal_power : 0;
k_temp_internal_power : 0;
k_volt_internal_power : 0;
k_process_rise_transition : 1;
k_temp_rise_transition : 0;
k_volt_rise_transition : 0;
k_process_fall_transition : 1;
k_temp_fall_transition : 0;
k_volt_fall_transition : 0;
k_process_setup_rise : 1;
k_temp_setup_rise : 0;
k_volt_setup_rise : 0;
k_process_setup_fall : 1;
k_temp_setup_fall : 0;
k_volt_setup_fall : 0;
k_process_hold_rise : 1;
k_temp_hold_rise : 0;
k_volt_hold_rise : 0;
k_process_hold_fall : 1;
k_temp_hold_fall : 0;
k_volt_hold_fall : 0;
k_process_min_pulse_width_high : 1;
k_temp_min_pulse_width_high : 0;
k_volt_min_pulse_width_high : 0;
k_process_min_pulse_width_low : 1;
k_temp_min_pulse_width_low : 0;
k_volt_min_pulse_width_low : 0;
k_process_recovery_rise : 1;
k_temp_recovery_rise : 0;
k_volt_recovery_rise : 0;
k_process_recovery_fall : 1;
k_temp_recovery_fall : 0;
k_volt_recovery_fall : 0;
k_process_cell_rise : 1;
k_temp_cell_rise : 0;
k_volt_cell_rise : 0;
k_process_cell_fall : 1;
k_temp_cell_fall : 0;
k_volt_cell_fall : 0;
k_process_wire_cap : 0;
k_temp_wire_cap : 0;
k_volt_wire_cap : 0;
k_process_wire_res : 0;
k_temp_wire_res : 0;
k_volt_wire_res : 0;
k_process_pin_cap : 0;
k_temp_pin_cap : 0;
k_volt_pin_cap : 0;

```

根据不同时序的计算需求，在时序信息文件中存在不同的比例因子定义。

4.1.3 输入电压组

输入电压组（input_voltage group）在库组中用于定义逻辑门单元输入端口的电压波动幅度，如下所示。

```
input_voltage(CMOS) {
    vil : 0.3 * VDD;
    vih : 0.7 * VDD;
    vmin : -0.5;
    vmax : VDD + 0.5;
}
```

输入电压组也可以用于定义输入 PAD 的电压波动幅度，如下所示。

```
input_voltage(TTL) {
    vil : 0.8;
    vih : 2;
    vmin : -0.5;
    vmax : VDD + 0.5;
}
```

4.1.4 输出电压组

输出电压组（output_voltage group）在库组中用于定义逻辑门单元输出端口的电压波动幅度，如下所示。

```
output_voltage(GENERAL) {
    vol : 0.4;
    voh : VDD - 0.4;
    vomin : -0.5;
    vomax : VDD + 0.5;
}
```

4.1.5 功耗查找表模板组

功耗查找表模板组（power_lut_template group）是库组中为其他功耗相关组所调用的查找表模板，通过该模板可以构成一维到三维的查找表结构，如下所示。

```
power_lut_template(name) {
    variable_1 : string ;
    variable_2 : string ;
    variable_3 : string ;
    index_1("float, ... , float") ;
    index_2("float, ... , float") ;
    index_3("float, ... , float") ;
}
```

如上所示存在 3 个不同的变量：variable_1、variable_2 和 variable_3，对应的变量参数值

也存在3种: total_output_net_capacitance、equal_or_opposite_output_net_capacitance 和 input_transition_time。通过3个索引变量 index_1、index_2 和 index_3 的值来确定最终的功耗结果。定义一个名为 energy_template_7x3x3 的功耗查找表模板组, 如下所示。

```
power_lut_template(energy_template_7x3x3) {
    variable_1 : input_transition_time;
    variable_2 : total_output_net_capacitance;
    variable_3 : equal_or_opposite_output_net_capacitance;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
    index_2 ("1000, 1001, 1002");
    index_3 ("1000, 1001, 1002");
}
```

4.1.6 操作条件组

操作条件组 (operating_conditions group) 是库组中用于定义时序分析时所对应的温度、电压、制程和 RC 网络模型的参数值。定义名为 fast 的操作条件组如下所示。

```
operating_conditions(fast) {
    process : 1;
    temperature : -40;
    voltage : 1.32;
    tree_type : balanced_tree
}
```

时序信息文件中可以定义多个不同名称的操作条件组。

4.1.7 线负载组

线负载组 (wire_load group) 是库组中用于定义时序分析时所使用的线负载模型, 时序分析工具基于线负载组中定义的单位电阻、单位电容和单位长度面积等参数值, 并通过具有扇出和缩放因子变量的公式来估算出线负载值。定义一个名为 fast_wl10 的线负载组如下所示。

```
wire_load("fast_wl10") {
    .....
}
```

在线负载组的内容里存在电阻属性 (resistance attribute)、电容属性 (capacitance attribute)、面积属性 (area attribute)、斜率属性 (slope attribute) 和扇出长度属性 (fanout_length attribute) 等, 这里只对几个主要的属性进行说明。

(1) 电阻属性

电阻属性定义互连线单位长度的电阻值, 该属性定义的示例如下。

```
resistance : 8.5e-8;
```

(2) 电容属性

电容属性定义互连线单位长度的电容值, 该属性定义的示例如下。

```
capacitance : 1.5e-4;
```

(3) 面积属性

面积属性定义互连线单位长度的面积值, 该属性定义的示例如下。

```
area : 0.7;
```

(4) 斜率属性

当互连线长度大于扇出长度属性中定义的最大长度时, 要基于斜率属性来线性地计算出超出的长度值。该属性定义的示例如下。

```
slope : 66.667;
```

(5) 扇出长度属性

扇出长度属性由扇出值和长度两个参数组成, 扇出值参数等于定义端口总数减去1, 长度参数通过统计的方法定义对应扇出值所得到的互连线面积值。该属性定义的示例如下。

```
fanout_length (1,66.667);
```

4.1.8 延时查找表模板组

延时查找表模板组 (lu_table_template group) 是库组中为其他延时相关组所调用的查找表模板, 通过该模板可以构成一维到三维的查找表结构, 如下所示。

```
lu_table_template(name) {
variable_1 : string ;
variable_2 : string ;
variable_3 : string ;
index_1("float, ... , float") ;
index_2("float, ... , float") ;
index_3("float, ... , float") ;
}
```

与功耗查找表模板组结构类似, 延时查找表模板组存在3个不同的变量: variable_1、variable_2 和 variable_3, 其中变量参数值根据所调用的延时相关组的不同而不同, 主要包括以下变量参数值: input_voltage、output_voltage、fanout_number、fanout_pin_capacitance、driver_slew、input_net_transition 和 total_output_net_capacitance 等, 通过3个索引变量 index_1、index_2 和 index_3 的值来确定最终的延时结果。定义一个名为 wire_delay_template_2x2x3 的延时查找表模板组如下所示。

```
lu_table_template(wire_delay_template_2x2x3) {
variable_1 : fanout_number;
variable_2 : fanout_pin_capacitance;
variable_3 : driver_slew;
index_1 ("1.0 , 3.0");
index_2 ("0.12, 4.24");
index_3 ("0.1, 2.7, 3.12");
}
```

4.1.9 单元组

库组中的单元组 (cell group) 是时序信息文件中定义逻辑门单元时序信息的组, 不同的逻辑门单元对应自己的单元组, 通过定义单元组的名称来进行区分。定义一个名为 AND2 的 2 输入与门单元组如下所示。

```
cell( AND2 ) {
    .....
}
```

在单元组的内容里存在面积属性 (area attribute)、单元标记属性 (cell_footprint attribute)、集成时钟门单元属性 (clock_gating_integrated_cell attribute)、不触碰属性 (dont_touch attribute)、不使用属性 (dont_use attribute) 和时钟门单元属性 (is_clock_gating_cell attribute) 等, 这里对主要的属性进行说明。

(1) 面积属性

面积属性定义单元的面积值, 一般只有 IO PAD 单元可以不定义面积属性值或者值为 0, 因为 IO PAD 单元不计入芯片内核面积中, 其他类型的单元应该定义面积属性值。该属性定义的示例如下。

```
area : 2.0;
```

(2) 单元标记属性

单元标记属性用于定义一类功能相同的单元, 单元标记属性相同的单元可以进行替换优化操作。该属性定义的示例如下。

```
cell_footprint : and2;
```

(3) 集成时钟门单元属性

集成时钟门单元属性用于定义一类单元为内部集成了时钟门的单元, 这样工具可以通过该属性识别出该集成时钟门的单元, 并用于低功耗设计。该属性定义的示例如下。

```
clock_gating_integrated_cell : "latch_posedge";
```

(4) 不触碰属性

不触碰属性用于定义对应的单元在工具对设计网表进行优化操作时维持原样, 这样可以使定义了不触碰属性的单元一直保留在设计网表中而不会被替换掉。该属性定义的示例如下。

```
dont_touch : true;
```

(5) 不使用属性

不使用属性用于定义对应的单元在工具对设计网表进行优化操作时禁止使用, 这样可以使定义了不使用属性的单元在工具进行优化操作时不会出现在设计网表中。该属性定义的示例如下。

```
dont_use : true;
```

(6) 时钟门单元属性

时钟门单元属性用于定义一类单元为时钟门的单元，这样工具可以通过该属性识别出该时钟门单元并用于低功耗设计。该属性定义的示例如下。

```
is_clock_gating_cell : true;
```

(7) 宏单元属性

宏单元属性 (is_macro_cell attribute) 用于定义一类单元为硬核单元 (或者黑匣子单元)。组成芯片的单元类型主要分为标准单元、宏单元和 IO PAD 单元，不同类型的单元在设计中的应用是有区别的，因此通过宏单元属性的定义可以使工具识别出宏单元，并进行相应的正确操作。该属性定义的示例如下。

```
is_macro_cell : true;
```

(8) 内存单元属性

内存单元属性 (is_memory_cell attribute) 用于定义一类单元为内存单元。通过内存单元属性的定义可以使工具识别出内存单元，并进行相应的正确操作。该属性定义的示例如下。

```
is_memory_cell : true;
```

(9) 仅映射属性

仅映射属性 (map_only attribute) 用于定义一类单元只在逻辑综合中使用，在逻辑优化中禁止使用。该属性定义的示例如下。

```
map_only : true;
```

(10) IO PAD 单元属性

IO PAD 单元属性 (pad_cell attribute) 用于定义一类单元为 IO PAD 单元。通过 IO PAD 单元属性的定义可以使工具识别出 IO PAD 单元，并进行相应的正确操作。该属性定义的示例如下。

```
pad_cell : true ;
```

(11) IO PAD 单元类型属性

IO PAD 单元类型属性 (pad_type attribute) 用于定义 IO PAD 单元的类型，不同类型的 IO PAD 对应不同的设计需求，比如用于时钟信号的 IO PAD、用于电源供电的 IO PAD 等。该属性定义的示例如下。

```
pad_type : clock ;
```

(12) 单元引脚名等价属性

单元引脚名等价属性 (pin_equal attribute) 用于定义不同的单元引脚名称代表相同的单元

引脚，这样可以把不同的单元引脚名称统一映射到相同的单元引脚上，使工具能够正确识别相同的单元引脚。该属性定义的示例如下。

```
pin_equal : ("Y Z") ;
```

(13) 单元引脚逻辑非属性

单元引脚逻辑非属性（`pin_opposite attribute`）用于定义逻辑成相反关系的一组单元引脚，使工具能够正确识别相应的单元引脚逻辑关系。该属性定义的示例如下。

```
pin_opposite("Q1 Q2 Q3", "QB1 QB2") ;
```

(14) 首选属性

首选属性（`preferred attribute`）用于定义在工具进行网表优化时优先使用的单元。该属性定义的示例如下。

```
preferred : true ;
```

(15) 缩放因子属性

缩放因子属性（`scaling_factors attribute`）用于定义一类单元不使用默认的缩放因子，而是使用缩放因子属性中定义的值。该属性定义的示例如下。

```
scaling_factors : IO_PAD_SCALING ;
```

4.1.10 引脚组

引脚组（`pin group`）是单元组中用于定义逻辑门单元端口时序信息的组，不同的逻辑门单元对应自己的引脚组。一个名为 AND2 的单元组中引脚 A 和引脚 B 的引脚组定义如下。

```
cell (AND) {
  area : 3 ;
  pin (A) {
    .....
  }
  pin (B) {
    .....
  }
}
```

在引脚组的内容里存在电容属性、带时钟门的时钟引脚属性（`clock_gate_clock_pin attribute`）、时钟门使能引脚属性（`clock_gate_enable_pin attribute`）和时钟门观察引脚属性（`clock_gate_obs_pin attribute`）等，这里对主要的属性进行说明。

(1) 电容属性

电容属性定义单元引脚的电容值，该属性定义的示例如下。

```
capacitance : 0.001400;
```

(2) 带时钟门的时钟引脚属性

带时钟门的时钟引脚属性定义与时钟门相关的时钟信号输入引脚，这样可以在综合和功耗分析时识别与时钟门信号相关的时钟引脚。该属性定义的示例如下。

```
clock_gate_clock_pin : true;
```

(3) 时钟门使能引脚属性

时钟门使能引脚属性定义实现时钟门功能的单元的时钟门使能信号的输入引脚。该属性定义的示例如下。

```
clock_gate_enable_pin : true;
```

(4) 时钟门观察引脚属性

时钟门观察引脚属性用于定义其单元引脚的输出结果为用于分析观察的信号。该属性定义的示例如下。

```
clock_gate_obs_pin : true;
```

(5) 时钟门输出引脚属性

时钟门输出引脚属性 (clock_gate_out_pin attribute) 用于定义由时钟门单元的控制的时钟信号输出引脚。该属性定义的示例如下。

```
clock_gate_out_pin : true;
```

(6) 时钟门测试模式引脚属性

时钟门测试模式引脚属性 (clock_gate_test_pin attribute) 定义该时钟门引脚用于连接测试或者扫描输入信号。该属性定义的示例如下。

```
clock_gate_test_pin : true;
```

(7) 方向属性

方向属性 (direction attribute) 用于定义单元引脚是输入、输出还是输入输出双向等方向特性。该属性定义的示例如下。

```
direction : output;
```

(8) 功能属性

功能属性 (function attribute) 定义时序单元的输出引脚逻辑状态信息，时序单元的输出引脚都需要定义功能属性。该属性定义的示例如下。

```
function : "IQN";
```

(9) 驱动类型属性

驱动类型属性 (driver_type attribute) 用于定义单元引脚的信号由哪种类型的单元所驱动，比如驱动信号来自上拉电阻、下拉电阻等。该属性定义的示例如下。


```
driver_type : pull_up;
```

(10) 模拟信号属性

模拟信号属性 (is_analog attribute) 用于定义单元引脚的信号为模拟信号。数字信号和模拟信号在实现要求上存在区别, 比如抗干扰要求、阻抗匹配要求等, 因此工具可以有效地识别模拟信号并进行正确的处理。该属性定义的示例如下。

```
is_analog : true;
```

(11) 扇出属性

扇出属性 (fanout_load attribute) 用于定义单元输入引脚的扇出值, 一般其值为 1, 如图 4-1 所示。

该属性定义的示例如下。

```
fanout_load : 1.0;
```

(12) 最大扇出属性

最大扇出属性 (max_fanout attribute) 用于定义单元输出引脚能够驱动的最大扇出值。该属性定义的示例如下。

```
max_fanout : 10;
```

(13) 最小扇出属性

最小扇出属性 (min_fanout attribute) 用于定义单元输出引脚能够驱动的最小扇出值。该属性定义的示例如下。

```
min_fanout : 2.0;
```

(14) 最大转换时间属性

最大转换时间属性 (max_transition attribute) 用于定义单元输入输出引脚能够接受的最大转换时间。该属性定义的示例如下。

```
max_transition : 0.764000;
```

(15) 最小转换时间属性

最小转换时间属性 (min_transition attribute) 用于定义单元输入输出引脚能够接受的最小转换时间。该属性定义的示例如下。

```
min_transition : 0.004000;
```

(16) 最大电容负载属性

最大电容负载属性 (max_capacitance attribute) 用于定义单元输出引脚能够接受的最大电容负载值。该属性定义的示例如下。

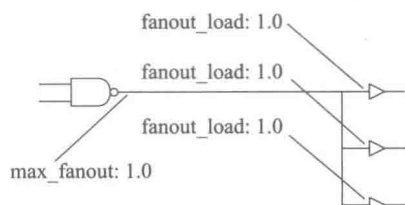


图 4-1 单元输入引脚扇出值

```
max_capacitance : 0.170640;
```

(17) 最小电容负载属性

最小电容负载属性 (min_capacitance attribute) 用于定义单元输出引脚能够接受的最小电容负载值。该属性定义的示例如下。

```
min_capacitance : 0.000040;
```

(18) 时钟属性

时钟属性 (clock attribute) 用于定义单元的引脚是否是用于连接时钟信号的引脚。该属性定义的示例如下。

```
clock : true;
```

(19) 最小时钟周期属性

最小时钟周期属性 (min_period attribute) 用于定义时序单元中接收时钟信号的引脚可以接受的时钟最小周期。该属性定义的示例如下。

```
min_period : 6.0;
```

(20) 最小高电平脉宽属性

最小高电平脉宽属性 (min_pulse_width_high attribute) 用于定义接收某些信号的单元引脚所能接受的最小高电平脉宽，这些信号可以是时钟信号、置位信号和清零信号等。该属性定义的示例如下。

```
min_pulse_width_high : 3;
```

(21) 最小低电平脉宽属性

最小低电平脉宽属性 (min_pulse_width_low attribute) 用于定义接收某些信号的单元引脚所能接受的最小低电平脉宽，这些信号可以是时钟信号、置位信号和清零信号等。该属性定义的示例如下。

```
min_pulse_width_low : 3;
```

4.1.11 触发器组

触发器组 (FF group) 是单元组中用于定义时序单元相关信息的组，时序单元类型可以是单级触发器也可以是主从触发器。一个名为 DFFX1 的单元组中触发器组定义如下所示。

```
cell (DFFX1) {
    .....
    ff ( IQ, IQN ) {
        .....
    }
}
```

在触发器组的内容里存有时钟有效属性 (clocked_on attribute)、下个状态属性 (next_state attribute)、清零属性 (clear attribute) 和置位属性 (preset attribute) 等, 这里对主要的属性进行说明。

(1) 时钟有效属性

时钟有效属性用于定义时序单元时钟输入引脚的名称及时钟有效边沿是正沿还是负沿。该属性定义的示例如下。

```
clocked_on : "CK";
```

(2) 下个状态属性

下个状态属性用于定义时序单元在对应输入引脚有效状态下保持的逻辑关系。该属性定义的示例如下。

```
next_state : "D";
```

(3) 清零属性

清零属性用于定义使时序单元储存值清零的输入引脚的名称及有效边沿。该属性定义的示例如下。

```
clear : "!RN";
```

(4) 置位属性

置位属性用于定义使时序单元储存值置位的输入引脚的名称及有效边沿。该属性定义的示例如下。

```
preset : "!SN";
```

4.1.12 逻辑状态表组

逻辑状态表组 (statetable group) 是单元组中用于定义复杂时序单元逻辑状态信息的组, 不同的复杂时序单元对应自己的逻辑状态表组。一个复杂时序单元的示意图如图 4-2 所示。

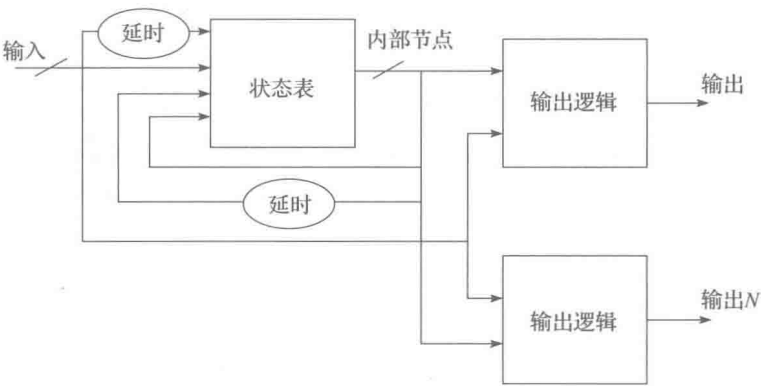


图 4-2 复杂时序单元的示意图

如图 4-2 所示，其中的逻辑状态表为用于描述输入与输出的逻辑关系真值表。
一个时序单元 DFFX1 的逻辑状态表组定义如下。

```
cell (DFFX1) {
.....
statetable( "CK E", "ENL") {
    table : "L L : - : L,\
            L H : - : H,\
            H - : - : N";
}
}
```

其中，L 表示低电平，H 表示高电平，N 表示无变化。

4.1.13 电源引脚组

电源引脚组 (pg_pin group) 是单元组中用于定义逻辑门单元电源和地引脚信息的组，当逻辑单元同时存在多组不同名称的电源地引脚时，可以通过电源引脚组来进行定义，这样工具能够正确识别逻辑单元不同的电源地引脚。不同的逻辑门单元对应自己的电源引脚组。一个名为 AND2 的单元组中名为 VDD 的电源引脚组定义如下。

```
cell (AND2) {
.....
pg_pin (VDD) {
.....
}
}
```

电源引脚组有电源地类型属性 (pg_type attribute) 和电源名称属性 (voltage_name attribute)。

(1) 电源地类型属性

电源地类型属性用于定义电源地是主干电源地还是非主干电源地，主干电源地设置为 primary_power 和 primary_ground，并且逻辑单元必须存在一组主干电源地设置。该属性定义的示例如下。

```
pg_type : "primary_power";
```

(2) 电源名称属性

电源名称属性用于定义电源的名称。该属性定义的示例如下。

```
voltage_name : "VDD";
```

4.1.14 延时组

延时组 (timing group) 是引脚组中用于定义逻辑门单元时序弧信息的组，不同的逻辑门单元对应自己的延时组。一个名为 AND2 的单元组中引脚 A 的延时组定义如下。

```
cell (AND2) {
    .....
    pin (A) {
        .....
        timing () {
            .....
        }
    }
}
```

在延时组的内容里存有相关引脚属性 (related_pin attribute)、延时逻辑状态属性 (timing_sense attribute) 和延时类型属性 (timing_type attribute) 等, 这里对主要属性进行说明。

(1) 相关引脚属性

相关引脚属性用于定义时序弧的始发点引脚。该属性定义的示例如下。

```
related_pin : "A";
```

(2) 延时逻辑状态属性

延时逻辑状态属性用于定义单元时序弧中输入引脚到输出引脚之间的信号变化逻辑关系。该属性定义的示例如下。

```
timing_sense : positive_unate;
```

(3) 延时类型属性

延时类型属性主要定义时序弧的类型, 根据单元功能逻辑可以把时序弧主要类型分为 4 大类: 组合逻辑 (combinational) 时序弧、时序逻辑 (sequential) 时序弧、非时序逻辑 (nonsequential) 时序弧和无变化 (no-change) 时序弧。通过延时类型属性, 工具可以正确识别组合逻辑单元和时序逻辑单元。如果没有设置该属性, 那么其默认设置是组合逻辑时序弧。该属性定义的示例如下。

```
timing_type : rising_edge;
```

4.1.15 单元上拉延时组

单元上拉延时组 (cell_rise group) 是延时组中用于定义逻辑门单元输出高电平的延时信息组, 不同的逻辑门单元对应自己的单元上拉延时组。由于单元上拉延时组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。一个 7×7 的查找表格式的单元上拉延时组定义如下所示。

```
timing () {
    .....
    cell_rise(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00632, 0.016432, 0.03792, 0.080896, 0.166848, 0.337488, 0.68256");
        values ( \
            "0.114696, 0.122629, 0.137074, 0.162152, 0.208483, 0.298242, 0.478976", \
```

```

"0.119774, 0.127673, 0.142094, 0.167181, 0.213495, 0.303266, 0.484002", \
"0.127199, 0.135194, 0.149580, 0.174619, 0.220974, 0.310726, 0.491454", \
"0.136372, 0.144175, 0.158513, 0.183506, 0.229793, 0.319567, 0.500314", \
"0.141040, 0.148751, 0.162809, 0.187516, 0.233620, 0.323392, 0.504123", \
"0.133187, 0.141040, 0.155083, 0.179601, 0.225464, 0.315025, 0.495759", \
"0.095823, 0.103607, 0.117803, 0.142592, 0.188293, 0.277490, 0.458158");
}
}

```

4.1.16 单元下拉延时组

单元下拉延时组 (cell_fall group) 是延时组中用于定义逻辑门单元输出低电平的延时信息组, 不同的逻辑门单元对应自己的单元下拉延时组。同样, 由于单元下拉延时组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。一个 7×7 的查找表格式的单元下拉延时组定义如下所示。

```

timing () {
.....
cell_fall(delay_template_7x7) {
    index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
    index_2 ("0.00632, 0.016432, 0.03792, 0.080896, 0.166848, 0.337488, 0.68256");
    values ( \
        "0.120273, 0.127716, 0.140800, 0.162053, 0.197340, 0.259756, 0.380629", \
        "0.125618, 0.133045, 0.146123, 0.167367, 0.202686, 0.265085, 0.385950", \
        "0.134291, 0.141727, 0.154824, 0.176112, 0.211462, 0.273871, 0.394737", \
        "0.148365, 0.155849, 0.169068, 0.190459, 0.225892, 0.288328, 0.409177", \
        "0.162665, 0.170548, 0.184290, 0.206208, 0.242086, 0.304745, 0.425576", \
        "0.166935, 0.175766, 0.190673, 0.213908, 0.250837, 0.314084, 0.434932", \
        "0.141368, 0.151510, 0.168595, 0.193963, 0.232479, 0.296763, 0.417905");
    )
}
}

```

4.1.17 上拉转换组

上拉转换组 (rise_transition group) 是延时组中用于定义逻辑门单元输出从低电平到高电平上拉转换的延时信息组, 不同的逻辑门单元对应自己的上拉转换组。同样, 由于上拉转换组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。一个 7×7 的查找表格式的上拉转换组定义如下所示。

```

timing () {
.....
rise_transition(delay_template_7x7) {
    index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
    index_2 ("0.00632, 0.016432, 0.03792, 0.080896, 0.166848, 0.337488, 0.68256");
    values ( \
        "0.032280, 0.041490, 0.061521, 0.100936, 0.183685, 0.353334, 0.699684", \
        "0.032132, 0.041890, 0.061651, 0.100740, 0.183609, 0.353371, 0.699685", \
        "0.032043, 0.041526, 0.061215, 0.100962, 0.183740, 0.353294, 0.699681", \

```

```

"0.030662, 0.040963, 0.061106, 0.100488, 0.183554, 0.353363, 0.699692", \
"0.031173, 0.041150, 0.060438, 0.099734, 0.183488, 0.353331, 0.699685", \
"0.030926, 0.041677, 0.059995, 0.099444, 0.182615, 0.353102, 0.699716", \
"0.032069, 0.042230, 0.061508, 0.099964, 0.181986, 0.352471, 0.699666");
}
}

```

4.1.18 下拉转换组

下拉转换组 (fall_transition group) 是延时组中用于定义逻辑门单元输出从高电平到低电平下拉转换的延时信息组, 不同的逻辑门单元对应自己的下拉转换组。同样, 由于下拉转换组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。一个 7×7 的查找表格式的下拉转换组定义如下所示。

```

timing () {
.....
fall_transition(delay_template_7x7) {
    index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
    index_2 ("0.00632, 0.016432, 0.03792, 0.080896, 0.166848, 0.337488, 0.68256");
    values ( \
        "0.033130, 0.040235, 0.055142, 0.081236, 0.130732, 0.232093, 0.446198", \
        "0.033304, 0.040384, 0.055198, 0.080952, 0.130688, 0.232038, 0.446179", \
        "0.033487, 0.040567, 0.055348, 0.081073, 0.130742, 0.232053, 0.446179", \
        "0.034927, 0.041996, 0.056490, 0.082166, 0.131123, 0.232173, 0.446173", \
        "0.039154, 0.046195, 0.060208, 0.085075, 0.132931, 0.232910, 0.446274", \
        "0.049607, 0.056437, 0.068668, 0.091213, 0.137340, 0.235026, 0.446713", \
        "0.061477, 0.068927, 0.081068, 0.101257, 0.143841, 0.238805, 0.447746");
    }
}
}

```

4.1.19 上拉约束组

上拉约束组 (rise_constraint group) 是延时组中用于定义时序单元输出高电平时的时序约束延时信息组, 即提供上拉约束时序弧信息的组, 不同的逻辑门单元对应自己的上拉约束组。同样, 由于上拉约束组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。一个 3×3 的查找表格式的上拉约束组定义如下所示。

```

timing () {
.....
rise_constraint(setup_template_3x3) {
    index_1 ("0.042, 0.392, 1.5");
    index_2 ("0.042, 0.392, 0.764");
    values ( \
        "0.074219, -0.023438, -0.101562", \
        "0.121094, 0.019531, -0.058594", \
        "0.179688, 0.078125, -0.003906");
    }
}
}

```

4.1.20 下拉约束组

下拉约束组 (fall_constraint group) 是延时组中用于定义时序单元输出低电平时的时序约束延时信息组, 即提供下拉约束时序弧信息的组, 不同的逻辑门单元对应自己的下拉约束组。同样, 由于下拉约束组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。一个 3×3 的查找表格式的下拉约束组定义如下所示。

```
timing () {
.....
fall_constraint(setup_template_3x3) {
    index_1 ("0.042, 0.392, 1.5");
    index_2 ("0.042, 0.392, 0.764");
    values ( \
        "0.316406, 0.230469, 0.167969", \
        "0.371094, 0.285156, 0.226562", \
        "0.582031, 0.496094, 0.433594");
    }
}
```

4.1.21 内部功耗组

内部功耗组 (internal_power group) 是引脚组中用于定义逻辑门单元功耗信息的组, 不同的逻辑门单元对应自己的内部功耗组。一个名为 AND2 的单元组中引脚 A 的内部功耗组定义如下所示。

```
cell (AND2) {
.....
pin (A) {
.....
internal_power() {
.....
}
}
}
```

在内部功耗组的内容里存在下拉功耗组 (fall_power group) 和上拉功耗组 (rise_power group)。

(1) 下拉功耗组

下拉功耗组是内部功耗组中用于定义单元输出低电平时的功耗信息组, 不同的逻辑门单元对应自己的下拉功耗组。同样, 由于下拉功耗组的数据格式是查找表的形式, 因此需要调用功耗查找表模板组来定义对应的查找表格式。一个 1×7 的查找表格式的下拉功耗组定义如下所示。

```
internal_power () {
fall_power(passive_energy_template_1x7) {
    index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
```



```

        values ("0.023418, 0.023391, 0.023400, 0.023478, 0.023890, 0.025581, 0.029964");
    }
}

```

(2) 上拉功耗组

上拉功耗组是内部功耗组中用于定义单元输出高电平时的功耗信息组，不同的逻辑门单元对应自己的上拉功耗组。同样，由于上拉功耗组的数据格式是查找表的形式，因此需要调用功耗查找表模板组来定义对应的查找表格式。一个 1×7 的查找表格式的上拉功耗组定义如下所示。

```

internal_power () {
    rise_power(passive_energy_template_1x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        values ("0.015343, 0.015357, 0.015380, 0.015510, 0.016192, 0.018124, 0.022579");
    }
}

```

4.1.22 哑阈漏流功耗组

哑阈漏流功耗组 (leakage_power group) 是单元组中用于定义单元自身由于哑阈漏流导致的功耗信息组，不同的逻辑门单元对应自己的哑阈漏流功耗组。一个名为 AND2 的单元组的哑阈漏流功耗组定义如下所示。

```

cell (AND2) {
    .....
    leakage_power() {
        .....
    }
}

```

在哑阈漏流功耗组的内容里存在模式属性 (mode attribute)、时刻属性 (when attribute) 和值属性 (value attribute)。

(1) 模式属性

模式属性用于定义单元在什么样的工作模式下有效，即可以调用哑阈漏流功耗组的信息。该属性定义的示例如下。

```
mode(rw, read);
```

(2) 时刻属性

时刻属性用于定义单元在执行特定的逻辑功能时刻下有效，即可以调用哑阈漏流功耗组的信息。该属性定义的示例如下。

```
when : "D & !CK & Q & !QN";
```

(3) 值属性

值属性用于定义单元的哑阈漏流功耗值。该属性定义的示例如下。

```
value : 68513.785200;
```

4.2 复合电流源延时模型

当时序模型 (CCS delay model) 是 CCS 延时模型时, 时序信息文件中就需要使用与 CCS 延时模型相关的组和属性。下面只介绍 CCS 延时模型中比较重要的组和属性。关于复合电流源延时模型文件的详细内容可以参考附录中 CCS 时序信息文件片段内容。

4.2.1 输出电流查找表模板组

输出电流查找表模板组 (output_current_template group) 是库组中为向量相关组所调用的查找表模板, 该模板为三维的查找表结构, 如下所示。

```
output_current_template(name) {
  variable_1: string;
  variable_2: string;
  variable_3: string;
  index_1("float, ... , float") ;
  index_2("float, ... , float") ;
  index_3("float, ... , float") ;
}
```

存在 3 种不同的变量: variable_1、variable_2 和 variable_3。对于调用该模板来表达的 CCS 延时模型的驱动模型, 其中变量参数值主要包括如下变量参数值: input_net_transition、total_output_net_capacitance 和 time, 通过 3 个索引变量 index_1、index_2 和 index_3 的值来确定最终的结果。如果索引变量的值不确定, 则可以留空, 根据以后调用该模板的具体组来确定。定义一个名为 CCS_TEMPLATE_0 的输出电流查找表模板组如下所示。

```
output_current_template ("CCS_TEMPLATE_0") {
  variable_1 : "input_net_transition";
  variable_2 : "total_output_net_capacitance";
  variable_3 : "time";
}
```

4.2.2 输出上拉电流组

输出上拉电流组 (output_current_rise group) 是延时组中用于定义 CCS 延时模型中单元输出高电平时电流信息的组, 不同的逻辑门单元对应自己的输出上拉电流组。其格式如下所示。

```
timing() {
  output_current_rise () {
    .....
  }
}
```

4.2.3 输出下拉电流组

输出下拉电流组 (output_current_fall group) 是延时组中用于定义 CCS 延时模型中单元输出低电平时电流信息的组, 不同的逻辑门单元对应自己的输出下拉电流组。其格式如下所示。

```
timing() {
  output_current_fall () {
    .....
  }
}
```

4.2.4 向量组

向量组 (vector group) 是输出上拉电流组和输出下拉电流组中用于储存电流值信息的组, 不同的逻辑门单元对应自己的向量组。由于向量组的数据格式是查找表的形式, 因此需要调用输出电流查找表模板组来定义对应的查找表格式。其格式如下所示。

```
output_current_rise() {
  vector(name) {
    .....
    index_1(float, ... , float);
    index_2(float, ... , float);
    index_3(float, ... , float);
    values(float, ... , float);
    .....
  }
}
```

根据不同的输入转换时间和输出负载, 向量组一一对应储存电流值信息, 因此会存在多个向量组并存的情况, 如下所示。

```
output_current_fall () {
  vector ("CCS_TEMPLATE_0") {
    index_1("10");
    index_2("1");
    index_3("12.49739, 17.72245, 23.35051, 30.23099, 42.13289");
    values("0.9, 0.7, 0.5, 0.3, 0.1");
  }
  vector ("CCS_TEMPLATE_0") {
    index_1("100");
    index_2("1");
    index_3("59.51487, 72.16286, 80.65922, 88.92921, 101.5645");
    values("0.9, 0.7, 0.5, 0.3, 0.1");
  }
}

output_current_rise () {
  vector ("CCS_TEMPLATE_0") {
    index_1("10");
```

```

        index_2("1");
        index_3("11.87746, 16.82628, 22.23928, 28.93928, 40.47045");
        values("0.1, 0.3, 0.5, 0.7, 0.9");
    }
    vector ("CCS_TEMPLATE_0") {
        index_1("100");
        index_2("1");
        index_3("55.05581, 67.5007, 75.96755, 84.33181, 96.89975");
        values("0.1, 0.3, 0.5, 0.7, 0.9");
    }
}

```

4.2.5 接收电容组

接收电容组 (receiver_capacitance group) 是引脚组中用于定义 CCS 延时模型中接收模型中电容信息的组, 不同的逻辑门单元对应自己的接收电容组。一个引脚 A 的接收电容组定义如下所示。

```

pin(A) {
    timing() {
        .....
        receiver_capacitance() {
            ...
        }
    }
}

```

在接收电容组内容里存在模式属性 (mode attribute) 和时刻属性 (when attribute)。

(1) 模式属性

模式属性用于定义单元在什么样的工作模式下有效, 即可以调用接收电容组的信息。该属性定义的示例如下。

```
mode (rw, write);
```

(2) 时刻属性

时刻属性用于定义单元在执行特定的逻辑功能时刻下有效, 即可以调用接收电容组的信息。该属性定义的示例如下。

```
when : "!CLK !D !RESET !Q";
```

前面介绍过, 由于复合电流源延时模型的接收模型由两个电容负载构成, 因此接收电容组中根据单元输出上拉和下拉状态, 上拉和下拉接收电容组一共分以下 4 组。

1. 上拉接收电容 1 组

上拉接收电容 1 组 (receiver_capacitance1_rise group) 是接收电容组中用于定义 CCS 延时模型中输出高电平时第一个接收模型中电容信息的组, 不同的逻辑门单元对应自己的上拉接收电容 1 组。由于上拉接收电容 1 组的数据格式是查找表的形式, 因此需要调用延时查找

表模板组来定义对应的查找表格式。其格式如下所示。

```
receiver_capacitance1_rise ("CCS_TEMPLATE") {
index_1("1.534715, 3.810094, 8.382257, 17.45951, 35.64224, 71.93259, 144.7454");
values("0.3754723, 0.4178715, 0.4443661, 0.4603675, 0.4641269, 0.466824, 0.465187");
}
```

2. 下拉接收电容 1 组

下拉接收电容 1 组 (receiver_capacitance1_fall group) 是接收电容组中用于定义 CCS 延时模型中输出低电平时第一个接收模型中电容信息的组, 不同的逻辑门单元对应自己的下拉接收电容 1 组。同样, 由于下拉接收电容 1 组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。其格式如下所示。

```
receiver_capacitance1_fall ("CCS_TEMPLATE") {
index_1("2.145208, 5.339855, 11.70128, 24.39093, 49.81852, 100.4779, 202.7041");
values("0.4981006, 0.4830799, 0.4805356, 0.4739592, 0.4633162, 0.4788486, 0.4769243");
}
```

3. 上拉接收电容 2 组

上拉接收电容 2 组 (receiver_capacitance2_rise group) 是接收电容组中用于定义 CCS 延时模型中输出高电平时第二个接收模型中电容信息的组, 不同的逻辑门单元对应自己的上拉接收电容 2 组。由于上拉接收电容 2 组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。其格式如下所示。

```
receiver_capacitance2_rise ("CCS_TEMPLATE") {
index_1("2.13896, 5.319148, 11.67641, 24.39093, 49.81997, 100.6781, 202.3943");
values("0.5357288, 0.5428979, 0.5185781, 0.4989055, 0.4942675, 0.4913622, 0.492515");
}
```

4. 下拉接收电容 2 组

下拉接收电容 2 组 (receiver_capacitance2_fall group) 是接收电容组中用于定义 CCS 延时模型中输出低电平时第二个接收模型中电容信息的组, 不同的逻辑门单元对应自己的下拉接收电容 2 组。同样, 由于下拉接收电容 2 组的数据格式是查找表的形式, 因此需要调用延时查找表模板组来定义对应的查找表格式。其格式如下。

```
receiver_capacitance2_fall ("CCS_TEMPLATE") {
index_1("1.529867, 3.791347, 8.343732, 17.51307, 35.65674, 72.41789, 144.3052");
values("0.3949011, 0.4394956, 0.4561169, 0.4667095, 0.4705957, 0.4703908, 0.4702269");
}
```

第 5 章

静态时序分析的基本方法

时序分析的目的是验证设计是否符合规定时序约束下的性能要求，同时设计者基于时序分析的结果来决定如何在不能满足时序要求的情况下进行时序性能改进。要想熟练地通过时序分析结果找到关键的时序违反路径并进行优化，就有必要了解时序分析的基本方法。

5.1 时序图

静态时序分析中，数字逻辑电路首先转换成时序图（timing graph）的形式，再基于时序图来计算路径延迟的总和，如果所有的路径都满足时序约束和规范，那么就认为设计满足时序约束规范。

生成时序图首先需要对要进行时序分析的设计进行逻辑电路到时序图的转换。如图 5-1 所示为一个需要进行时序分析的逻辑电路图。

时序图主要包括时序路径的逻辑节点、连接关系和主要输入输出端口。逻辑节点为设计中的逻辑单元，连接关系为设计中逻辑单元的输入输出关系，主要输入输出端口为设计的输入输出端口。如图 5-2 所示为逻辑电路图与时序图转换前的中间结果。

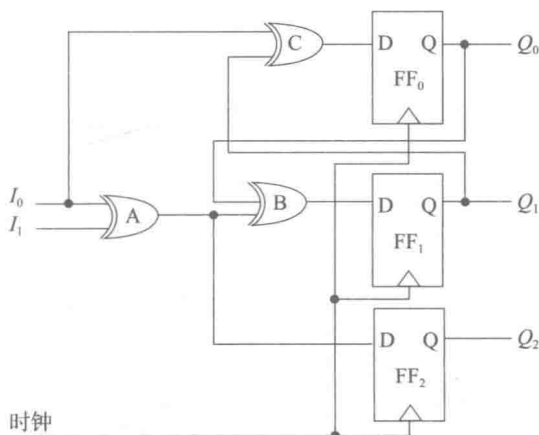


图 5-1 逻辑电路图

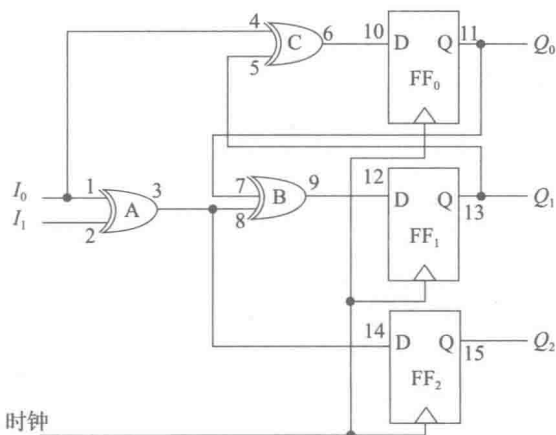


图 5-2 时序图中间结果

由于时序分析基于4种类型的时序路径进行分析,所以时序图中起点和终点也根据如下原则进行定义:

1) 时序图中起点主要包括设计边界上数据输入端口、时序单元(flip flop)的数据输出端口和存储器(memory)数据输出端口这3类。

2) 时序图中终点包括设计边界上数据输出端口、时序单元的数据输入端口和存储器数据输入端口这3类。

基于以上原则,以上逻辑电路图转换成的时序图如图5-3所示。

如图5-3所示,逻辑电路图转换成时序图的起点应该有5个,分别为 I_0 、 I_1 、11、13和15。同样逻辑电路图转换成时序图的终点应该有6个,分别为 Q_0 、 Q_1 、 Q_2 、10、12和14。

完成基本的电路逻辑图到时序图转换后,还需要结合连接关系、时序弧、时序约束、时序分析策略和计算方法等其他相关因素,才能对设计进行时序分析。

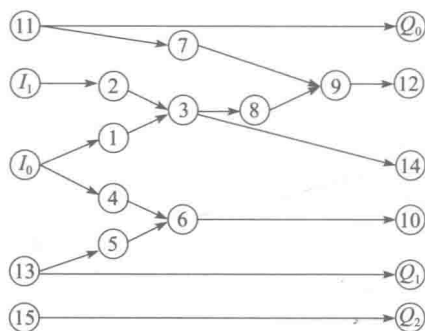


图 5-3 时序图转换结果

5.2 时序分析策略

对设计进行时序分析前,需要选择合适的时序分析策略。时序分析策略主要分为两种:基于路径(path-based)的时序分析策略和基于模块(block-based)的时序分析策略。

1. 基于路径的时序分析策略

基于路径的时序分析策略是基于时序图找到从起点到终点的所有路径并进行时序分析,如图5-4所示。

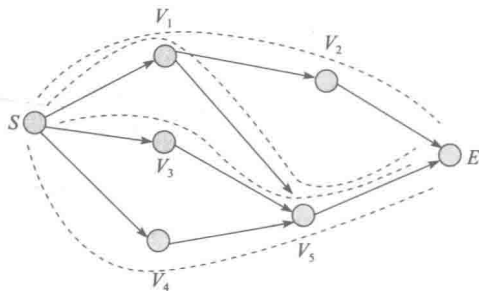


图 5-4 基于路径的时序分析策略

如图5-4所示,该时序图的结构,通过基于路径的时序分析策略会独立地分析计算虚线标记的4条时序路径。

基于路径的时序分析策略能够独立精确地计算每条时序路径,但是随着设计规模的增加,这种时序分析策略需要基于单元延时和连线延时等诸多信息来计算时序图里所有时序路径,这样可能增加计算量和运行时间,然而可以通过合理设置来屏蔽不必要时序路径的时序分析。主流的时序分析工具一般基于该时序分析策略进行时序分析,如PrimeTime。

2. 基于模块的时序分析策略

基于模块的时序分析策略也可以称为基于图的时序分析策略(graph-based),它是基于时序图找到从起点到终点的所有路径,在进行时序分析时只基于该路径下延迟最严重的节点进

行计算,如图 5-5 所示。

通过以上时序图的结构,同时结合节点的延时信息,基于模块的时序分析策略会对所有时序路径的节点进行分析,从而分析出最关键的时序路径并进行时序计算。因此基于模块的时序分析策略只计算由虚线标记的一条时序路径。

基于模块的时序分析策略能够同时分析所有时序路径,并且随着设计规模的增加,计算量和运行时间不会像基于路径的分析策略那样大幅增加。但是其缺点是其分析精度没有基于路径的分析略高。

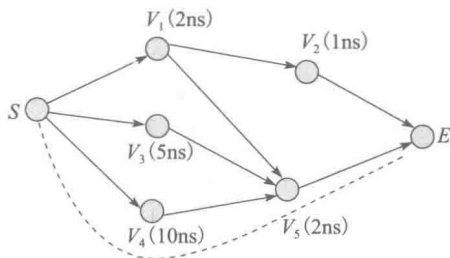


图 5-5 基于模块的时序分析策略

5.3 时序路径延时计算方法

当与设计相关的所有时序弧都存在并有效时,时序路径延时就可以基于获取的时序弧信息并运用合适的算法来进行计算。时序路径延时的计算根据如下几种逻辑组合形式而有不同的计算方法。

1. 组合逻辑之间路径延时计算方法

组合逻辑电路在逻辑功能上的特点是:任意时刻的输出仅仅取决于该时刻的输入,与电路原来的状态无关。最基本的组合逻辑之间的延时计算方法是通过把逻辑路径的逻辑门延时和信号线延时逐一相加来实现的。

下面介绍计算一条以 3 个反相器单元组成的组合逻辑路径延时的基本方法。其组合逻辑路径如图 5-6 所示。

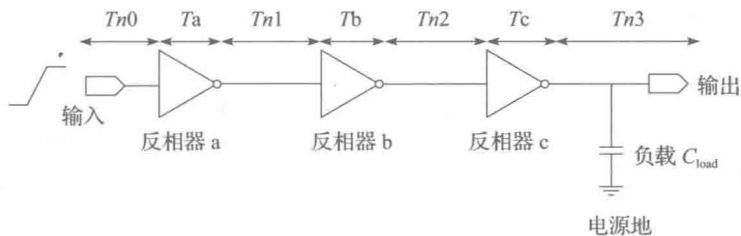


图 5-6 纯组合逻辑路径

默认情况下,时序路径的输入端信号转换时间的值为 0,同时输出端的负载电容值 (C_{load}) 也为 0。但是默认值并不符合实际情况,因此为了符合实际的电路工作环境,输入端 N_0 的信号转换时间需要人为根据设计实际情况进行定义,这样通过输入端 N_0 的信号转换时间来计算出输入端的线延时和第一个反向器单元的实际自身延时。同样,输出端 N_3 的负载电容值也需要人工定义,这样通过输出端 N_3 的负载电容值计算出输出端线延时和最后一个

反相器单元的实际自身延时。

从 N_0 点到 N_3 点的组合逻辑路径延时计算需要考虑输入上拉和下拉这两个不同状态下的组合逻辑路径延时。由于反相器单元是具有逻辑功能的单元，因此根据输入的不同其对应的时序弧也不同。

输入信号为上拉状态的计算公式如式 (5-1) 所示。

$$T_{\text{rise}} = Tn0_{\text{fall}} + Ta_{\text{rise}} + Tn1_{\text{rise}} + Tb_{\text{fall}} + Tn2_{\text{fall}} + Tc_{\text{rise}} + Tn3_{\text{rise}} \quad (5-1)$$

输入信号为下拉状态的计算公式如式 (5-2) 所示。

$$T_{\text{fall}} = Tn0_{\text{rise}} + Ta_{\text{fall}} + Tn1_{\text{fall}} + Tb_{\text{rise}} + Tn2_{\text{rise}} + Tc_{\text{fall}} + Tn3_{\text{fall}} \quad (5-2)$$

其中, $T_{\text{rise/fall}}$ 为输入端信号为上拉和下拉时的组合逻辑总延时, $Tn0_{\text{rise/fall}}$ 、 $Tn1_{\text{rise/fall}}$ 、 $Tn2_{\text{rise/fall}}$ 和 $Tn3_{\text{rise/fall}}$ 为对应输入状态的线延时值, $Ta_{\text{rise/fall}}$ 、 $Tb_{\text{rise/fall}}$ 和线 $Tc_{\text{rise/fall}}$ 为反相器单元在对应输入状态下的自身单元延时。

2. 时序逻辑之间路径延时计算方法

时序逻辑电路在逻辑功能上的特点是: 任意时刻的输出不仅取决于当时的输入信号, 而且还取决于电路原来的状态。时序逻辑之间最基本的延时计算方法也是通过把逻辑路径的逻辑门延时和信号线延时逐一相加来实现的。但是时序逻辑单元只在时钟有效沿时工作, 因此需要分别计算时序逻辑单元的时钟信号路径延时和数据信号路径延时。

时序逻辑之间路径是始发点和终止点都为时序单元的时序路径, 始发点时序单元和终止点时序单元的时钟信号分为两种: 始发点时序单元的时钟称为发射时钟 (launch clock), 终止点时序单元的时钟称为捕获时钟 (capture clock), 如图 5-7 所示。

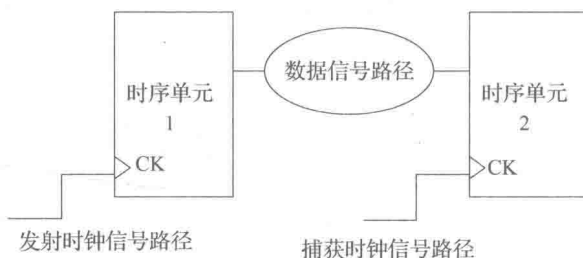


图 5-7 发射时钟和捕获时钟

因此计算时序逻辑单元的时钟信号路径延时需要分别计算捕获时钟路径和发射时钟路径。

下面介绍计算一条时序逻辑之间路径延时基本方法, 其时序逻辑路径如图 5-8 所示。

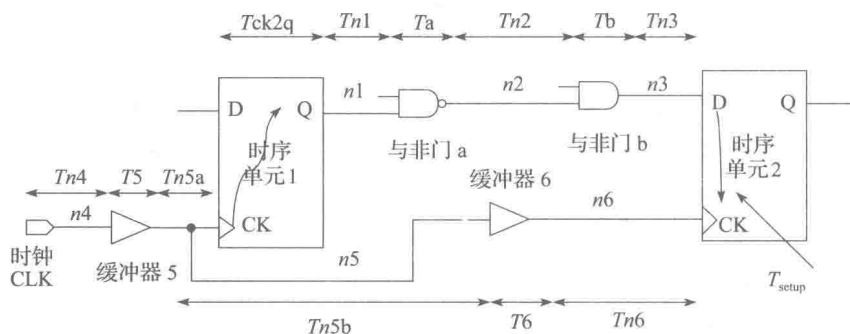


图 5-8 包含时序单元的时序逻辑路径

前面已经介绍过不同的输入信号状态决定使用对应状态下的时序弧进行延时计算, 这里以其中一个输入状态下的情况来进行说明。

数据信号路径延时计算如式(5-3)所示。

$$T_{ck2q_{rise}} + T_{n1_{rise}} + T_{a_{fall}} + T_{n2_{fall}} + T_{b_{fall}} + T_{n3_{fall}} \quad (5-3)$$

其中, $T_{ck2q_{rise}}$ 为始发点时序单元自身的单元延时, $T_{n1_{rise}}$ 、 $T_{n2_{fall}}$ 和 $T_{n3_{fall}}$ 为数据信号路径上的线延时值, $T_{a_{fall}}$ 和 $T_{b_{fall}}$ 为组合逻辑单元在对应输入状态下的自身单元延时。

发射时钟信号路径延时计算如式(5-4)所示。

$$T_{n4_{rise}} + T_{5_{rise}} + T_{n5a_{rise}} \quad (5-4)$$

捕获时钟信号路径延时计算如式(5-5)所示。

$$T_{n4_{rise}} + T_{5_{rise}} + T_{n5b_{rise}} + T_{6_{rise}} + T_{n6_{rise}} \quad (5-5)$$

其中, $T_{n4_{rise}}$ 、 $T_{5_{rise}}$ 和 $T_{n5b_{rise}}$ 为时钟信号路径上的线延时值, $T_{5_{rise}}$ 和 $T_{n6_{rise}}$ 为组合逻辑单元在对应输入状态下的自身单元延时。从式(5-5)中可以看出, 捕获时钟路径与发射时钟存在公共路径, 并且捕获时钟逻辑较长, 因此其计算公式中要计算的单元延时变量相应也较多。

5.4 时序路径的分析方法

对时序路径的建立时间(setup time)和保持时间(hold)进行分析是静态时序分析的两个主要工作。

1. 建立时间分析(setup timing check)

建立时间是指在时钟捕获信号有效沿到来前, 数据信号必须提前达到稳定状态的时间。

这里以最典型的时序路径类型触发器到触发器为例, 如图5-9所示。

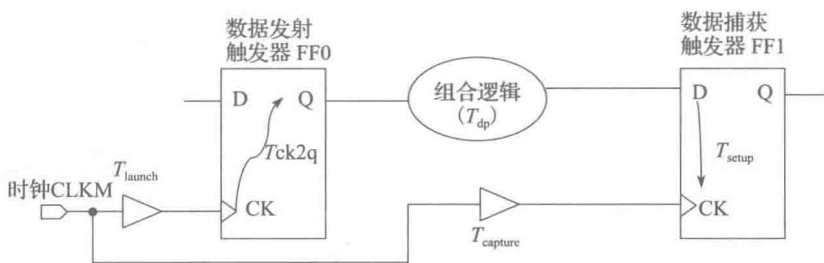


图 5-9 建立时间分析

始发点触发器也称为数据发射触发器(launch flip-flop), 其数据输入端口D端的数据在时钟信号有效沿的触发下, 经过组合逻辑(Combinational logic), 到达终止点触发器(也称为数据捕获触发器(capture flip-flop))的数据输入端口D端。当下一个时钟有效沿到来时, 对终止点触发器进行建立时间的检查。假设时钟有效信号CLKM从时钟源点到达始发点触发器时钟端口CK的时间为 T_{launch} , 时钟端口到Q的时间为 T_{ck2q} , 中间组合逻辑传播延时为 T_{dp} , 时

钟有效信号为 CLKM 从时钟源点到达终止点触发器时钟输入端口 CK 端的时间为 T_{capture} ，终止点触发器的建立时间为 T_{setup} 。为保证建立时间的满足，由始发点触发器输出的数据必须在下一个时钟有效沿到来前就已经达到稳定，其时序波形图如图 5-10 所示。

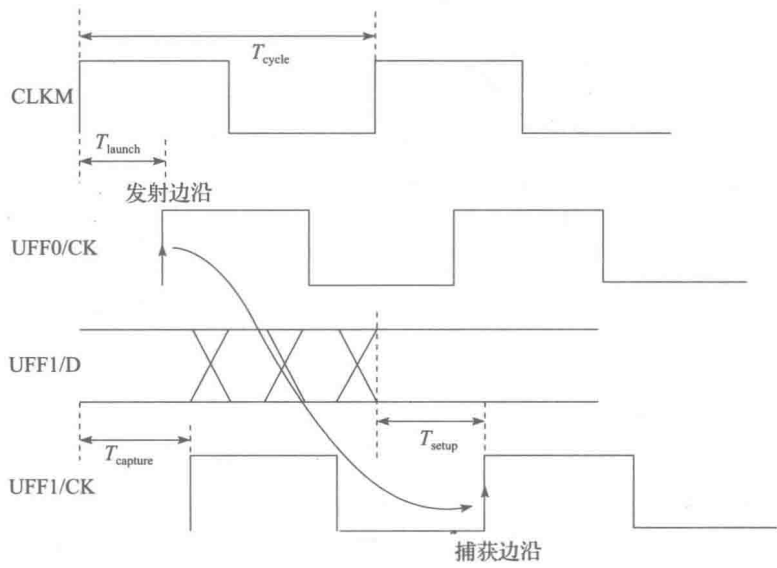


图 5-10 建立时间分析时序波形图

该类型时序路径的建立时间必须满足式 (5-6)：

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{\text{dp}} < T_{\text{capture}} + T_{\text{cycle}} - T_{\text{setup}} \quad (5-6)$$

因此，时序路径的建立时间分析是验证时序路径中时钟与数据之间时序关系是否满足终止点时序单元建立时间的要求。

2. 保持时间分析 (hold timing check)

保持时间是指为保证时序单元对数据读取正确，数据在时钟有效沿到来之后仍需要保持稳定的时间。同样，这里以最典型的时序路径类型触发器到触发器为例，如图 5-11 所示。

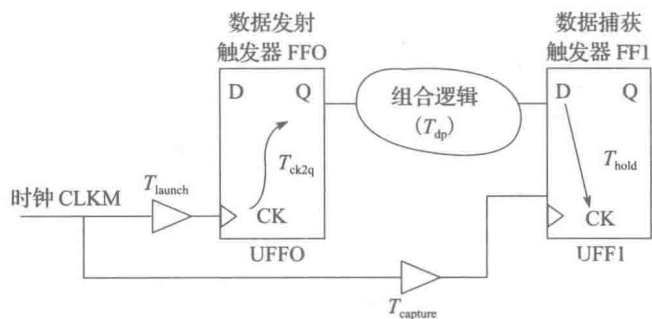


图 5-11 保持时间分析

始发点触发器数据端的数据在时钟信号有效沿的触发下, 经过组合逻辑到达下一个时序单元终止点触发器的数据输入端口, 当下一个时钟有效沿到来时, 对终止点触发器进行保持时间的检查。与建立时间检查类似, 假设时钟有效信号 CLKM 从时钟源点到始发点触发器时钟端口 CK 的时间为 T_{launch} , 时钟端口到 Q 的时间为 T_{ck2q} , 中间组合逻辑经历的时间为 T_{dp} , 时钟有效信号为 CLKM 从时钟源点到达终止点触发器 CK 端的时间为 T_{capture} , 终止点触发器的保持时间为 T_{hold} 。为保证保持时间的满足, 由终止点触发器接收的数据, 必须在捕获时钟有效沿之后继续维持一段稳定的时间, 其时序波形图如图 5-12 所示。

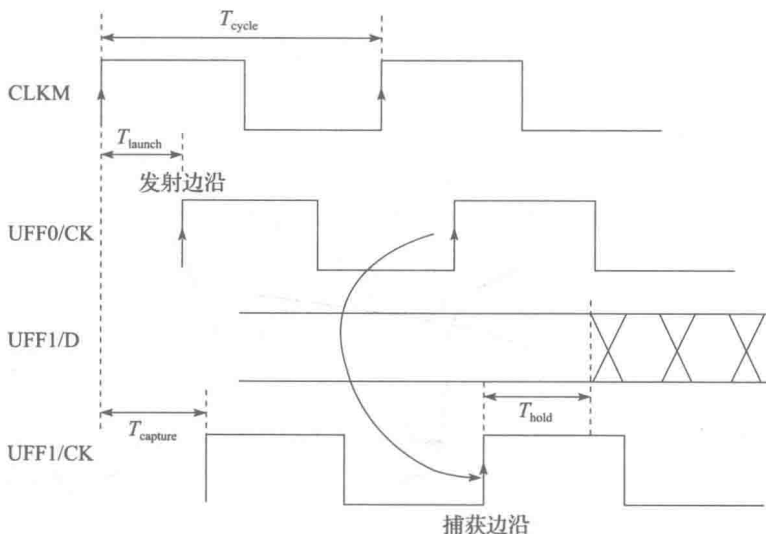


图 5-12 保持时间分析时序波形图

因此该类型时序路径的保持时间必须满足式 (5-7)。

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{\text{dp}} > T_{\text{capture}} + T_{\text{hold}} \quad (5-7)$$

时序路径的保持时间分析是验证时序路径中时钟与数据之间时序关系是否满足终止点时序单元保持时间的要求。

前面已经介绍过, 时序路径根据始发点和终止点的不同分为 4 种类型的时序路径: 触发器到触发器、触发器到输出端、输入端到触发器和输入端到输出端。同时时序路径分析又分为建立时间分析和保持时间分析两种类型, 因此对应不同的类型的时序路径存在对应的时序路径分析方法。

1. 触发器到触发器

时钟为理想情况下, 发射时钟路径延时和捕获时钟路径延时都为 0。当存在实际的时钟网络时, 需要考虑实际的发射时钟路径延时和捕获时钟路径延时。

该时序路径计算分析基本方法如图 5-13 所示。

存在实际的时钟网络时建立时间要求, 如式 (5-8) 所示。

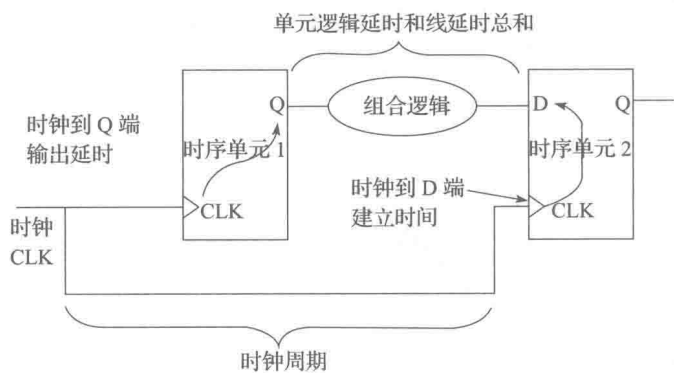


图 5-13 触发器到触发器时序路径分析方法

发射时钟路径延时 + 始发点时序单元 CLK → Q 延时 + 组合逻辑路径延时
≤ 捕获时钟路径延时 + 时钟周期 - 终止点时序单元建立时间 (5-8)

存在实际的时钟网络时保持时间要求，如式 (5-9) 所示。

发射时钟路径延时 + 始发点时序单元 CLK → Q 延时 + 组合逻辑路径延时
≥ 捕获时钟路径延时 + 终止点时序单元保持时间 (5-9)

2. 触发器到输出端

同样，时钟为理想情况下，发射时钟路径延时和捕获时钟路径延时都为 0。当存在实际的时钟网络时，需要考虑实际的发射时钟路径延时和捕获时钟路径延时，该类型时序路径的捕获时钟传播延时通过设置虚拟时钟传播延时来实现。

该时序路径计算分析基本方法如图 5-14 所示。

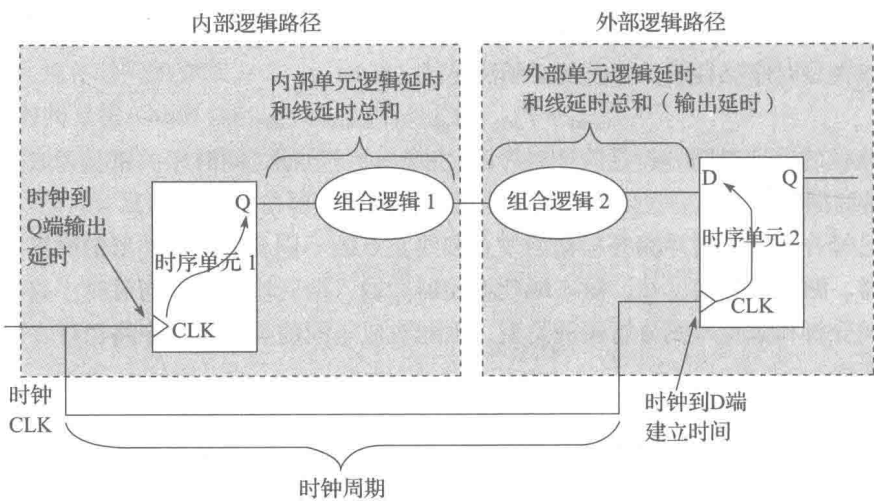


图 5-14 触发器到输出端时序路径分析方法

存在实际时钟网络时建立时间要求，如式 (5-10) 所示。

发射时钟路径延时 + 始发点时序单元 CLK → Q 延时 + 组合逻辑路径延时

$$\leq \text{虚拟捕获时钟路径延时} + \text{时钟周期} - \text{输出延时} \quad (5-10)$$

存在实际的时钟网络时保持时间要求, 如式 (5-11) 所示。

发射时钟路径延时 + 始发点时序单元 CLK → Q 延时 + 组合逻辑路径延时

$$\geq \text{虚拟捕获时钟路径延时} + \text{时序单元保持时间}(0) - \text{输出延时} \quad (5-11)$$

3. 输入端到触发器

同理, 时钟为理想情况下, 发射时钟路径延时和捕获时钟路径延时都为 0。当存在实际的时钟网络时, 需要考虑实际的发射时钟路径延时和捕获时钟路径延时, 该类型时序路径的发射时钟传播延时通过设置虚拟时钟传播延时来实现。

该时序路径计算分析基本方法如图 5-15 所示。

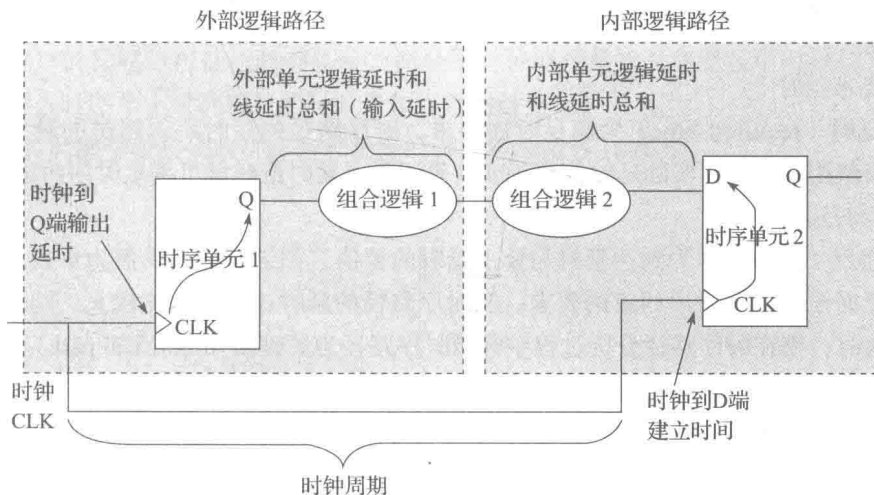


图 5-15 输入端到触发器时序路径分析方法

存在实际的时钟网络时建立时间要求, 如式 (5-12) 所示。

$$\text{虚拟发射时钟路径延时} + \text{输入延时} + \text{组合逻辑路径延时} \leq \text{捕获时钟路径} + \text{时钟周期} - \text{终止点时序单元建立时间} \quad (5-12)$$

存在实际的时钟网络时保持时间要求, 如式 (5-13) 所示。

$$\text{虚拟发射时钟路径延时} + \text{输入延时} + \text{组合逻辑路径延时} \geq \text{捕获时钟路径} + \text{终止点时序单元保持时间} \quad (5-13)$$

4. 输入端到输出端

在这种情况下, 因为没有经过任何的时序单元, 所以没有可以作为参考的时钟信号。根据具体设计需求, 可以通过设置虚拟时钟等方法对这种时序路径进行时序分析。由于虚拟时钟为理想时钟, 因此对应的虚拟发射时钟路径延时和虚拟捕获时钟路径延时都为 0,

如图 5-16 所示。

建立时间要求，如式 (5-14) 所示。

$$\text{组合逻辑路径延时} \leq \text{虚拟时钟周期} - \text{输入时间} - \text{输出时间} \quad (5-14)$$

保持时间要求，如式 (5-15) 所示。

$$\text{组合逻辑路径延时} \geq \text{时序单元保持时间 (0)} - \text{输入时间} - \text{输出时间} \quad (5-15)$$

时序路径分析的目的就是检查时序路径的传播延时是否满足时序约束的要求，因此是否满足时序约束要求是基于如下 3 个重要参数来进行说明。

(1) 到达延时

到达延时 (arrival time) 是指该时序路径实际的传播延时，其延时是根据具体的线延时、单元延时等信息并根据时序路径延时计算方法计算而来。

(2) 要求延时

要求延时 (required time) 为满足时序要求，时序路径的延时不得超过的最大延时极限值，该极限值用来说明当实际延时值小于或者等于要求延时值时即可满足时序约束。

(3) 延时违反

延时违反 (slack) 即为到达延时与要求延时的差值，如果延时违反值为负数，则说明时序路径的延时没有满足时序约束的要求。当时序路径的延时违反数值比较大，同时时序修复难度比较大时，则在时序路径分析过程中称该时序路径为关键路径 (critical path)。

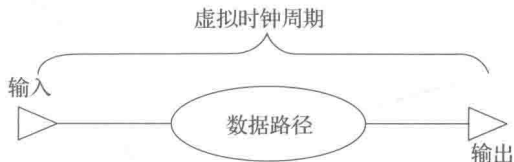


图 5-16 输入端到输出端时序路径分析方法

5.5 时序路径分析模式

在详细说明时序路径分析模式之前，先介绍时序路径分析模式中所涉及的两种计算时序路径类型：最快路径和最慢路径。时序路径分析模式中需要选择正确的最快路径和最慢路径进行时序计算。

1. 最快路径

最快路径 (early path) 指在信号传播延时计算中调用最快工艺参数的路径，根据信号的分类可以分为最快时钟路径和最快数据路径。

2. 最慢路径

最慢路径 (late path) 指在信号传播延时计算中调用最慢工艺参数的路径，分为最慢时钟路径和最慢数据路径。

与数据路径不同，最快时钟路径、最慢时钟路径的选择在建立时间分析和保持时间分析中是不同的。

1) 建立时间分析最快时钟路径和最慢时钟路径如图 5-17 所示。

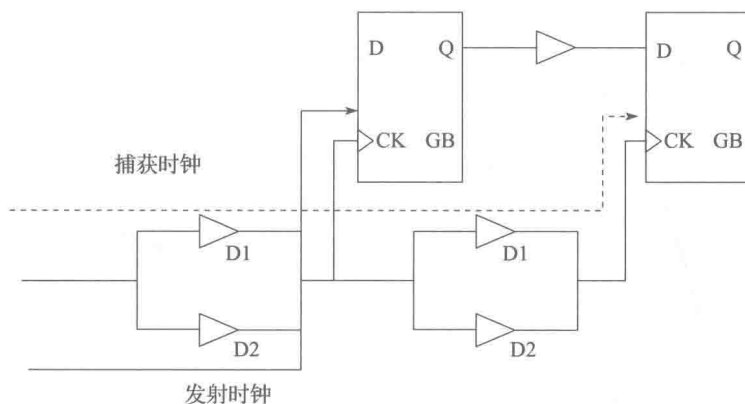


图 5-17 建立时间中 fastest 时钟路径和最慢时钟路径

在建立时间分析中，最快时钟路径是指时序路径中时钟信号从时钟源点到达终止点时序单元时钟端口的延时最短捕获时钟路径，而最慢时钟路径是指时序路径中时钟信号从时钟源点到达始发点时序单元时钟端口的延时最长发射时钟路径。

2) 保持时间分析 fastest 时钟路径和最慢时钟路径如图 5-18 所示。

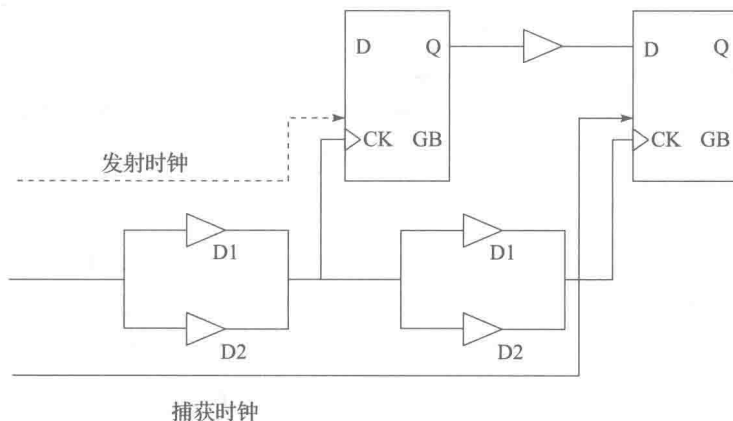


图 5-18 保持时间中 fastest 时钟路径和最慢时钟路径

在保持时间分析中，最快时钟路径是指时序路径中时钟信号从时钟源点到达始发点时序单元时钟端口的延时最短发射时钟路径，而最慢时钟路径是指时序路径中时钟信号从时钟源点到达终止点时序单元时钟端口的延时最长捕获时钟路径。

静态时序分析工具提供 3 种分析模式进行静态时序分析，不同的设计需求通过选择对应的时序分析模式，从而可以在合理的时序计算负荷范围内得到接近于实际工作的时序分析结果。这 3 种时序路径分析模式分别为：单一分析模式 (single mode)、最好 - 最坏分析模式 (BC-WC mode)、全芯片变化分析模式 (OCV mode)。

5.5.1 单一分析模式

前面介绍过 PVT 环境分 3 类：最好的、典型的和最坏的工作环境。最好对应较小的工艺偏差、较低的工作温度以及较高的工作电压，最坏对应较大的工艺偏差、较高的工作温度和较低的工作电压，典型则是上述两种极端条件的折中。对于单一分析模式，静态时序分析工具只会在指定的一种工作条件下检查建立时间和保持时间。

根据前面所介绍的触发器到触发器时序路径建立时间要求，转换成单一分析模式下建立时间的基本计算如式（5-16）所示：

$$\begin{aligned} & \text{发射时钟最慢路径延时} + \text{最慢数据路径延时} \leq \text{捕获时钟最快路径延时} \\ & \quad + \text{时钟周期} - \text{终止点时序单元建立时间} \end{aligned} \quad (5-16)$$

进行建立时间检查时，始发点触发器的发射时钟路径延时、终止点触发器捕获时钟路径沿和从始发点到终止点的数据路径延时都是基于单一工作条件下所计算的路径延时。

这里以最典型的时序路径类型触发器到触发器为例，进行单一分析模式下建立时间计算，如图 5-19 所示。

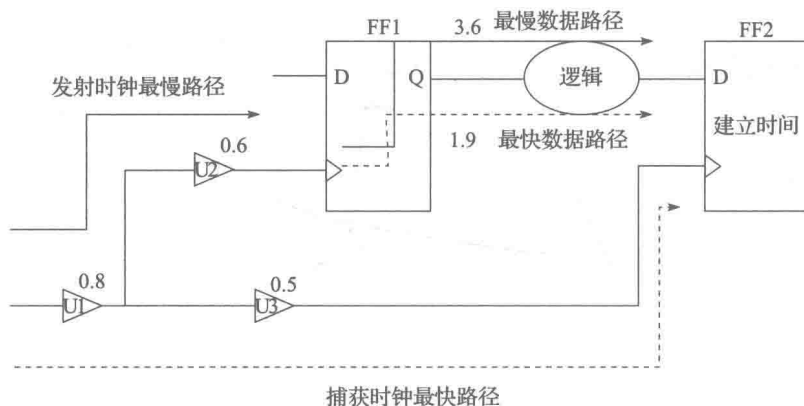


图 5-19 单一分析模式下建立时间计算（单位：ns）

根据图 5-19 中的延时参数，单一分析模式下建立时间计算结果如下：

时钟周期 = 4

发射时钟最慢路径延时值 = U1 单元延时 + U2 单元延时 = 0.8 + 0.6 = 1.4

最慢数据路径延时值 = 3.6

最快数据路径延时值 = 1.9

捕获时钟最快路径延时值 = U1 单元延时 + U3 单元延时 = 0.8 + 0.5 = 1.3

时序单元的建立时间要求值 = 0.2

数据到达延时值 = 发射时钟最慢路径延时值 + 最慢数据路径延时值 = 1.4 + 3.6 = 5

数据要求延时值 = 时钟周期 + 捕获时钟最快路径延时值 - 时序单元的建立时间要求值
= 4 + 1.3 - 0.2 = 5.1

延时违反值 = 数据要求延时值 - 数据到达延时值 = $5.1 - 5 = 0.1$

根据前面所介绍的触发器到触发器时序路径保持时间要求，并转换成单一分析模式下保持时间的基本计算式 (5-17) 如下。

$$\begin{aligned} & \text{发射时钟最快路径延时} + \text{最快数据路径延时} \geq \text{捕获时钟最慢路径延时} \\ & \quad + \text{终止点时序单元保持时间} \end{aligned} \quad (5-17)$$

同样，进行保持时间检查始发点触发器的发射时钟路径延时、终止点触发器捕获时钟路径延时和从始发点到终止点的数据路径延时与建立时间检查一样，都是基于相同的单一工作条件下所计算的路径延时。

这里以最典型的时序路径类型触发器到触发器为例，进行单一分析模式下保持时间计算，如图 5-20 所示。

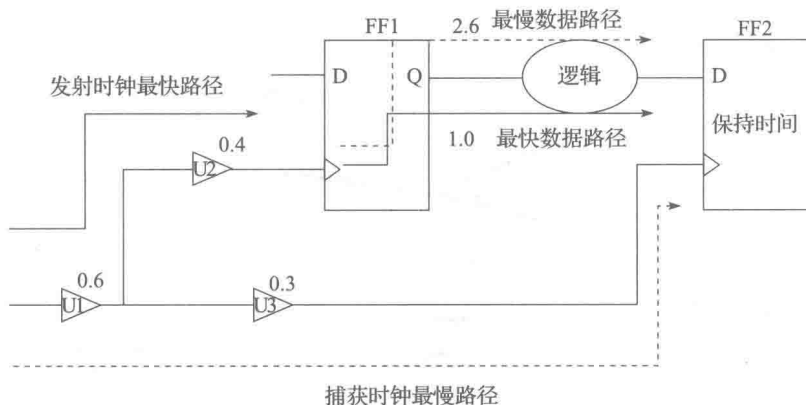


图 5-20 单一分析模式下保持时间计算 (单位: ns)

根据图 5-20 中的延时参数，单一分析模式下保持时间计算结果如下：

时钟周期 = 4

发射时钟最快路径延时值 = U1 单元延时 + U2 单元延时 = $0.6 + 0.4 = 1.0$

最快数据路径延时值 = 1.0

最慢数据路径延时值 = 2.6

捕获时钟最慢路径延时值 = U1 单元延时 + U3 单元延时 = $0.6 + 0.3 = 0.9$

时序单元的保持时间要求值 = 0.1

数据到达延时值 = 发射时钟最快路径延时值 + 最快数据路径延时值 = $1.0 + 1.0 = 2.0$

数据要求延时值 = 时序单元的保持时间要求值 + 捕获时钟最慢路径延时值 = $0.1 + 0.9 = 1.0$

延时违反值 = 数据要求延时值 - 数据到达延时值 = $1.0 - 2.0 = -1.0$

5.5.2 最好 - 最坏分析模式

对于最好 - 最坏分析模式，静态时序分析工具会同时在 PVT 环境中的最好的和最坏的

工作环境下检查建立时间和保持时间。

最好-最坏分析模式中建立时间的基本计算公式与单一分析模式下建立时间的基本计算公式一致，不同点在于计算建立时间所使用的工作环境不同，在计算建立时间过程中静态时序分析工具调用逻辑单元的最大(Max)延时时序库，并用来检查时序路径最大延时是否满足触发器建立时间的约束。

进行建立时间检查时，始发点触发器的发射时钟延时、终止点触发器捕获时钟延时和从始发点到终止点的数据路径延时都是基于最坏工作条件下所计算的路径延时。

这里以最典型的时序路径类型触发器到触发器为例，进行最好-最坏分析模式下建立时间计算，如图 5-21 所示。

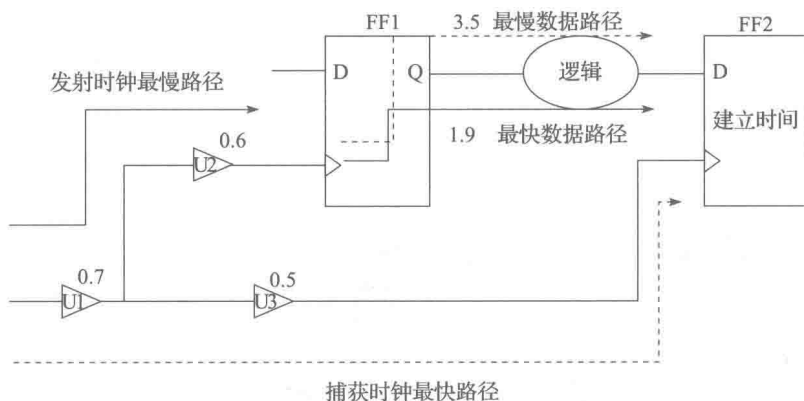


图 5-21 最好-最坏分析模式下建立时间计算 (单位: ns)

根据图 5-21 中的延时参数，最好-最坏分析模式下建立时间计算结果如下：

时钟周期 = 4

发射时钟最慢路径延时值 (Max 库) = U1 单元延时 (Max 库) + U2 单元延时 (Max 库)
 $= 0.7 + 0.6 = 1.3$

最慢数据路径延时值 (Max 库) = 3.5

最快数据路径延时值 (Max 库) = 1.9

捕获时钟最快路径延时值 (Max 库) = U1 单元延时 (Max 库) + U3 单元延时 (Max 库)
 $= 0.7 + 0.5 = 1.2$

时序单元的建立时间要求值 (Max 库) = 0.2

数据到达延时值 = 发射时钟最慢路径延时值 (Max 库) + 最慢数据路径延时值 (Max 库)
 $= 1.3 + 3.5 = 4.8$

数据要求延时值 = 时钟周期 + 捕获时钟最快路径延时值 (Max 库)
 $-$ 时序单元的建立时间要求值 (Max 库) = $4 + 1.2 - 0.2 = 5.0$

延时违反值 = 数据要求延时值 - 数据到达延时值 = $5.0 - 4.8 = 0.2$

同样, 最好-最坏路径分析模式中保持时间的基本计算公式与单一分析模式下保持时间的基本计算公式一致, 不同点在于计算保持时间所使用的工作环境不同, 在计算保持时间过程中静态时序分析工具调用逻辑单元的最小 (Min) 延时时序库, 并用来检查时序路径最小延时是否满足触发器保持时间的约束。

进行保持时间检查时, 始发点触发器的发射时钟延时、终止点触发器捕获时钟延时和从始发点到终止点的数据路径延时都是基于最好工作条件下所计算的路径延时。

这里以最典型的时序路径类型触发器到触发器为例, 进行最好-最坏分析模式下保持时间计算, 如图 5-22 所示。

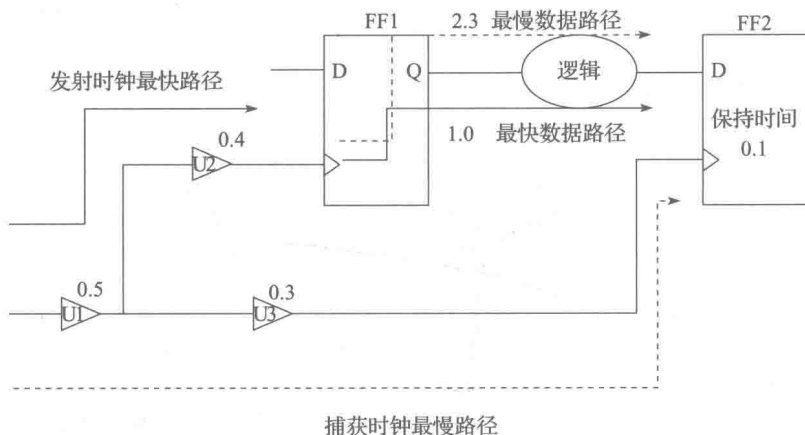


图 5-22 最好-最坏分析模式下保持时间计算 (单位: ns)

根据图 5-22 中的延时参数, 最好-最坏分析模式下保持时间计算结果如下:

时钟周期 = 4

发射时钟最快路径延时值 (Min 库) = U1 单元延时 (Min 库) + U2 单元延时 (Min 库)
 $= 0.5 + 0.4 = 0.9$

最快数据路径延时值 (Min 库) = 1.0

最慢数据路径延时值 (Min 库) = 2.3

捕获时钟最慢路径延时值 (Min 库) = U1 单元延时 (Min 库) + U3 单元延时 (Min 库)
 $= 0.5 + 0.3 = 0.8$

时序单元的保持时间要求值 (Min 库) = 0.1

数据到达延时值 = 发射时钟最快路径延时值 (Min 库) + 最快数据路径延时值 (Min 库)
 $= 0.9 + 1.0 = 1.9$

数据要求延时值 = 时序单元的保持时间要求值 (Min 库)
 $+ \text{捕获时钟最慢路径延时值 (Min 库)} = 0.1 + 0.8 = 0.9$

延时违反值 = 数据要求延时值 - 数据到达延时值 = $0.9 - 1.9 = -1.0$

5.5.3 芯片变化相关分析模式

由于工艺偏差的存在，导致不同晶圆之间、同一晶圆不同芯片之间、同一芯片不同区域之间的流片情况都是不相同的。造成这些区别的因素有很多种，这些因素造成的区别主要有几个体现：电压降造成局部不同的供电的差异；晶体管阈值电压的差异；晶体管沟道长度的差异；局部热点形成的温度系数的差异；互连线不同引起的电阻电容的差异。

芯片变化相关分析模式是可以用来更加严谨苛刻的方法来检测建立时间以及保持时间，使得时序分析结果更加接近于目前先进制造工艺下真实情况的一种分析手段。OCV 分析模式可以描述 PVT 对单个芯片所造成的影响，还可以同时用来考虑长距离走线对时钟路径的影响。

在芯片变化相关工作模式下，与最好 - 最坏分析模式一样，静态时序分析工具也会同时在 PVT 环境中的最好的和最坏的工作环境下检查建立时间和保持时间。

OCV 分析模式中建立时间的基本计算公式与其他分析模式下建立时间的基本计算公式一致，不同点在于计算最快路径和最慢路径所使用的工作环境不同，在计算建立时间过程中静态时序分析工具调用逻辑单元的最大延时时序库来计算最慢路径的延时，同时调用逻辑单元的最小延时时序库来计算最快路径的延时，来检查时序路径的延时是否满足触发器建立时间的约束。

进行建立时间检查时，始发点触发器的发射时钟采用的是最坏条件下最慢时钟路径，终止点触发器的捕获时钟采用的是最好条件下最快时钟路径，而从始发点到终止点的数据路径的延时则是在最坏条件下最慢数据路径延时。

这里以最典型的时序路径类型触发器到触发器为例，进行 OCV 模式下建立时间的计算，如图 5-23 所示。

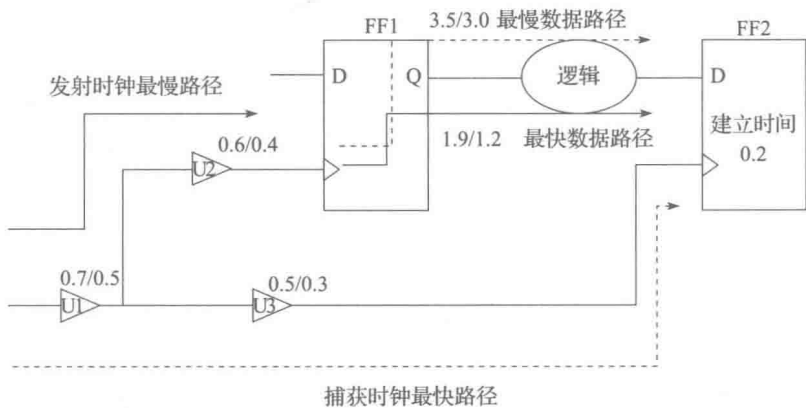


图 5-23 OCV 模式下建立时间计算 (单位: ns)

根据图 5-23 中的延时参数，OCV 分析模式下建立时间计算结果如下：

时钟周期 = 4

发射时钟最慢路径延时值 (Max 库) = U1 单元延时 (Max 库) + U2 单元延时 (Max 库)
= 0.7 + 0.6 = 1.3

最慢数据路径延时值 (Max 库) = 3.5

最快数据路径延时值 (Min 库) = 1.2

捕获时钟最快路径延时值 (Min 库) = U1 单元延时 (Min 库) + U3 单元延时 (Min 库)
 $= 0.5 + 0.3 = 0.8$

时序单元的建立时间要求值 (Max 库) = 0.2

数据到达延时值 = 发射时钟最慢路径延时值 (Max 库) + 最慢数据路径延时值 (Max 库)
 $= 1.3 + 3.5 = 4.8$

数据要求延时值 = 时钟周期 + 捕获时钟最快路径延时值 (Min 库)
 $-$ 时序单元的建立时间要求值 (Max 库) = $4 + 0.8 - 0.2 = 4.6$

延时违反值 = 数据要求延时值 - 数据到达延时值 = $4.6 - 4.8 = -0.2$

同样, OCV 分析模式中保持时间的基本计算公式与其他分析模式下保持时间的基本计算公式一致, 不同点同样在于计算最快路径和最慢路径所使用的工作环境不同, 在计算建立时间过程中静态时序分析工具调用逻辑单元的最大延时时序库来计算最慢路径的延时, 调用逻辑单元的最小延时时序库来计算最快路径的延时来检查时序路径的延时是否满足触发器保持时间的约束。OCV 分析模式下计算保持时间与建立时间不同点在于所确定的最快路径和最慢路径发生了变化。

进行保持时间检查时, 始发点触发器的发射时钟采用的是最好条件下最快时钟路径, 终止点触发器的捕获时钟采用的是最坏条件下最慢时钟路径, 从始发点到终止点的数据路径延时则是在最好条件下最快数据路径延时。

这里以最典型的时序路径类型触发器到触发器为例, 进行 OCV 分析模式下保持时间计算, 如图 5-24 所示。

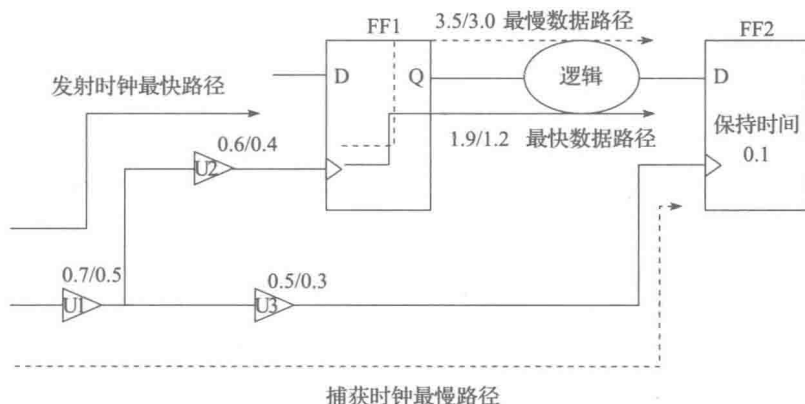


图 5-24 OCV 分析模式下保持时间计算 (单位: ns)

根据图 5-24 中的延时参数, OCV 分析模式下保持时间计算结果如下:
 时钟周期 = 4

发射时钟最快路径延时值 (Min 库) = U1 单元延时 (Min 库) + U2 单元延时 (Min 库)
= 0.5 + 0.4 = 0.9

最快数据路径延时值 (Min 库) = 1.2

最慢数据路径延时值 (Max 库) = 3.5

捕获时钟最慢路径延时值 (Max 库) = U1 单元延时 (Max 库) + U3 单元延时 (Max 库)
= 0.7 + 0.5 = 1.2

时序单元的保持时间要求值 (Max 库) = 0.1

数据到达延时值 = 发射时钟最快时钟路径延时值 (Min 库) + 最快数据路径延时值 (Min 库)
= 0.9 + 1.2 = 2.1

数据要求延时值 = 时序单元的保持时间要求值 (Max 库)
+ 捕获时钟最慢路径延时值 (Max) = 0.1 + 1.2 = 1.3

延时违反值 = 数据要求延时值 - 数据到达延时值 = 2.1 - 1.3 = 0.8

5.6 时序减免

时序减免 (timing derate) 的作用是根据减免 (derating) 系数, 静态时序分析工具会在时序路径的每级逻辑门、连线 and 端口上都加上或减去一个原来延时值乘以减免系数值的延时作为最终的延时结果。设置时序减免值的目的是使时序分析结果更加符合实际情况。静态时序分析工具通过使用命令 `set_timing_derate` 来定义时序减免值。

这里以一条典型的时序路径为例, 来介绍建立时间分析中时序减免的作用, 如图 5-25 所示。

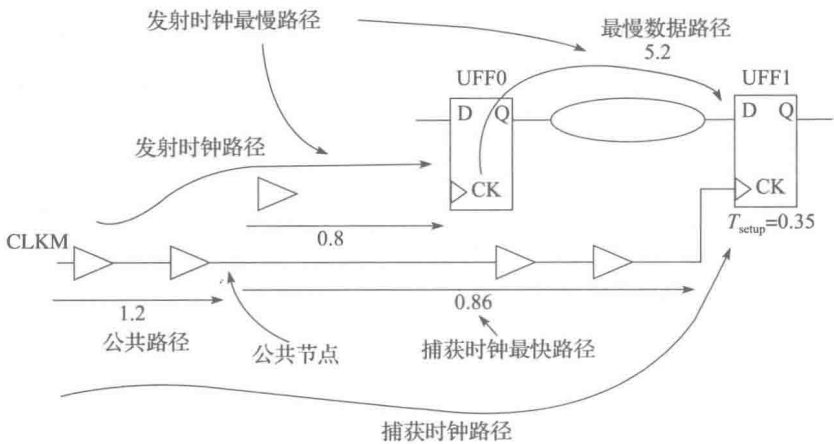


图 5-25 时序减免的计算 (单位: ns)

建立时间的基本计算公式如式 (5-18) 所示。

发射时钟路径延时 + 最慢数据路径延时 \leq 时钟周期 + 捕获时钟路径延时

- 终止点时序单元建立时间要求值 (5-18)

当没有考虑设置时序减免值时, 在 OCV 模式下, 图 5-25 中建立时间分析结果如下:

发射时钟最慢路径延时 = $1.2 + 0.8 = 2.0$

最慢数据路径延时 = 5.2

捕获时钟最快路径延时 = $1.2 + 0.86 = 2.06$

终止点时序单元建立时间要求值 = 0.35

为满足时序要求的最小时钟周期如下。

发射时钟最慢路径延时 + 最慢数据路径延时 - 捕获时钟最快

路径延时 - 终止点时序单元建立时间要求值

= $2.0 + 5.2 - 2.06 + 0.35 = 5.49$

当设置时序减免值后, 其设置如下。

```
set_timing_derate -early 0.9
```

```
set_timing_derate -late 1.2
```

以上时序减免设置定义最快路径延时值在原基础值上再减少 10%, 最慢路径延时值在原基础值上再增加 20%。

基于以上设置的减免值, 在 OCV 模式下, 图 5-25 中建立时间分析结果如下:

发射时钟最慢路径延时 = $2.0 \times 1.2 = 2.4$

最慢数据路径延时 = $5.2 \times 1.2 = 6.24$

捕获时钟最快路径延时 = $2.06 \times 0.9 = 1.854$

终止点时序单元建立时间要求值 = 0.35

为满足时序要求的最小时钟周期如下:

发射时钟最慢路径延时 + 最慢数据路径延时 - 捕获时钟最快

路径延时 - 终止点时序单元建立时间要求值

= $2.4 + 6.24 - 1.854 + 0.35 = 7.136$

当芯片进入深亚微米设计时, 工艺制造过程中的偏差对芯片时序的影响越来越大, 芯片变化相关分析模式已被广泛应用于静态时序分析中。但是实际应用芯片变化相关分析模式时, 通常不会同时读入最好和最坏两个时序库。因为随着生产工艺越来越先进, 实际设计所使用的工艺情况 (corner) 已经远远大于 3 种, 并且最好和最坏时序库之间的分析结果差距太大。如果进行建立时间检查时, 始发点的发射时钟使用最坏时序库, 而终止点的捕获时钟使用最好时序库, 最后 OCV 模式分析下的时序结果会过于悲观。同样, 如果使用最好和最坏两套时序库来分析保持时间, 结果也会过于悲观。因此在实际静态时序分析中, 一次只读入一套时序库, 并通过设置减免值的方法来达到进行 OCV 模式分析的目的。

对于基于单独一套库下的 OCV 分析情况, 为实现类似最好和最坏时序库的效果, 设计中通过设置 derate 值来实现, 如下所示。


```
set_timing_derate -early 0.9
set_timing_derate -late 1.1
```

通过以上减免值的设置，静态时序分析工具在计算时序延时中，最快路径（early path）会基于单一时序库计算结果再减少 10% 的延时值，这类似于调用了最好时序库来计算最快路径的效果。同样最慢路径（late path）会基于单一时序库计算结果再增加 10% 的延时值，这类似于调用了最坏时序库来计算最慢路径的效果。在实际工程设计中，对于不同的生产工艺，时序减免的设定值是通过实际工程经验总结出来的。

5.7 其他芯片变化相关分析模式

随着制造工艺越来越先进，在时序分析规模不断增大同时，对时序分析精度的要求也越来越高，因此常规的芯片变化相关分析模式已经无法满足当前更高级的时序分析要求。

本节将介绍两种更高级的芯片变化相关分析模式：高级芯片变化相关分析模式（advanced on-chip variation mode）和参数化芯片变化相关分析模式（parametric on-chip variation mode）

1. 高级芯片变化相关分析模式

在真实状态下，由于晶圆的片上局部工艺偏差，PVT 等因素在单个芯片所造成的影响是随机的，比如在同一条时序路径上，可能有的单元的延时会加快，有的单元的延时会变慢。基于常规的芯片变化相关分析模式的时序分析方法由于使用统一的时序减免值，所以该分析模式存在缺点。基于 OCV 分析模式的建立时间计算示意图如图 5-26 所示。

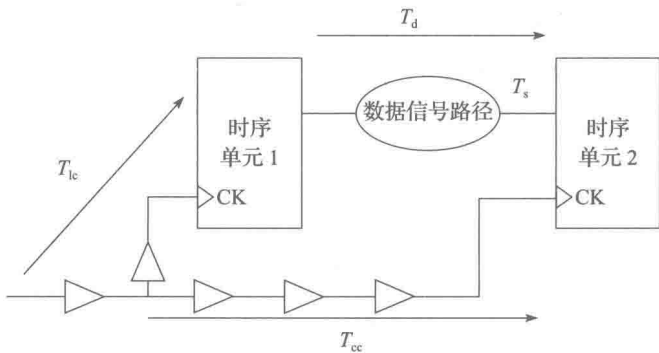


图 5-26 OCV 分析模式的建立时间计算

如图 5-26 所示，建立时间的计算式（5-19）如下。

$$T_{1c} + T_d + T_s < T_{cc} + T_p \tag{5-19}$$

其中， T_{1c} 为发射时钟最快路径延时值， T_d 为最快数据路径延时值， T_s 为时序单元 2 的建立时间要求值， T_{cc} 为最慢捕获时钟路径延时值， T_p 为时钟周期。

时序减免值设置如下：

```
set_timing_derate -early 0.9
set_timing_derate -late 1.1
```

那么 T_{lc} 、 T_d 、 T_s 的延时会变大, T_{cc} 的延时会变快。其建立时间由于在最快路径和最慢路径中分别使用统一的时序减免值, 这样最快路径延时越来越快, 最慢路径延时越来越慢, 使时序分析结果过于悲观, 这样会导致时序收敛难度加大。

OCV 分析模式在保持时间的计算上也同样存在时序分析过于悲观的问题, 如图 5-27 所示。

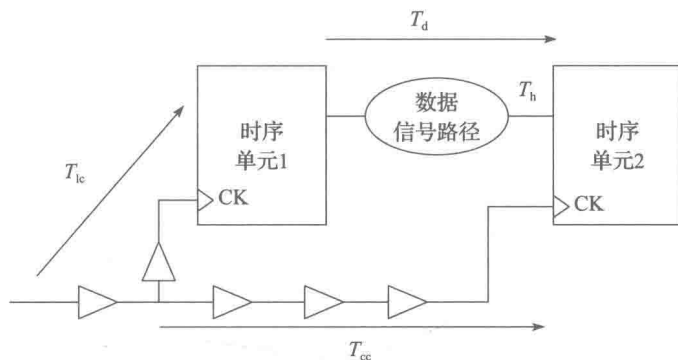


图 5-27 OCV 分析模式的保持时间计算

如图 5-27 所示, 保持时间的计算式 (5-20) 如下。

$$T_{lc} + T_d > T_{cc} + T_h \quad (5-20)$$

其中, T_{lc} 为发射时钟最快路径延时值, T_d 为最快数据路径延时, T_h 为时序单元 2 的保持时间要求值, T_{cc} 为最慢捕获时钟路径延时。

时序减免值设置如下:

```
set_timing_derate -early 0.9
set_timing_derate -late 1.1
```

那么 T_{lc} 、 T_d 的延时会变快, T_{cc} 、 T_h 的延时会变慢。和计算建立时间一样, 其保持时间分析在计算中由于在最快路径和最慢路径中分别使用统一的时序减免值, 这样最快路径延时越来越快, 最慢路径延时越来越慢, 那使时序分析结果过于悲观, 导致时序收敛难度加大。

高级芯片变化相关分析模式简称 AOCV 分析模式, 它通过更详细的模拟仿真结果来最大程度地接近真实情况下片上局部变化影响的随机性, 通过动态地调整时序减免值来达到更加接近真实情况的目的。因此高级芯片变化相关分析模式可以使时序分析结果更加接近真实状态, 从而减少悲观结果带来的时序收敛难度。

由于局部片上随机的工艺偏差 (random on-die variation) 影响会随着时序路径的逻辑长度的增加而减小, 同时有规律的工艺偏差 (systematic on-die variation) 影响也会随着时序路径所跨越的物理距离的增加而增加, 因此 AOCV 分析模式分别通过分析时序路径的逻辑深度和物理距离, 分别覆盖局部片上随机的工艺偏差和有规律的工艺偏差的影响, 来动态调用对

应的时序减免值进行时序分析计算。AOCV 分析模式动态地调整时序减免值是通过查找表的方法来实现的, AOCV 分析模式查找表格式如下。

```
version: 1.0
object_type: design
rf_type: rise fall
delay_type: cell net
path_type: clock data
derate_type: early
object_spec: top
depth: 0 1 2 3
distance: 100 200
table: 0.87 0.93 0.95 0.96 \
0.83 0.85 0.87 0.90
```

如上所示, AOCV 分析模式查找表格式中存在如下几个重要参数: 版本 (version)、对象类型 (object_type)、上拉下拉类型 (rf_type)、延时类型 (delay_type)、路径类型 (path_type)、减免路径类型 (derate_type)、对象属性 (object_spec)、深度 (depth)、距离 (distance) 和查找表 (table)。

(1) 版本

由于不同版本存在格式、内容和分析模式的区别, 因此版本参数用来定义该 AOCV 查找表数据的版本。

(2) 对象类型

对象类型参数定义 AOCV 用于哪种类型的对象上, 对象分为 3 种: design、lib_cell 和 cell。

(3) 上拉下拉类型

上拉下拉类型参数用于定义 AOCV 有效的电平转换状态, 电平转换状态分为 3 种: rise, fall 和 rise fall。

(4) 延时类型

由于时序分析主要由单元延时和互连线延时组成, 因此延时类型参数用于定义延时计算时 AOCV 对哪种延时类型有效, 延时对象分为 3 种: cell、net 和 cell net。

(5) 路径类型

由于时序分析主要由时钟路径和数据路径组成, 因此路径类型参数用于定义延时计算时 AOCV 对哪种路径类型有效, 延时对象分为 3 种: clock、data 和 clock data。

(6) 减免路径类型

由于 AOCV 分析中存在最快路径和最慢路径, 因此减免路径类型参数用于定义 AOCV 有效时所对应的最快路径或者最慢路径, 减免路径类型分为两种: early 和 late。

(7) 对象属性

根据对象类型参数的定义, 需要定义对应的对象属性参数, 该参数用于定义 AOCV 作用于对象的模式。

(8) 深度

深度参数定义时序路径的逻辑长度值,通过深度值可以在 AOCV 查找表中找到对应的时序减免值。

(9) 距离

距离参数定义时序路径的物理距离值,通过距离值也可以在 AOCV 查找表中找到对应的时序减免值。

(10) 查找表

查找表中定义了不同的时序减免值,通过索引参数深度和距离来确定对应的时序减免值。一个 AOCV 分析模式在保持时间计算的例子如图 5-28 所示。

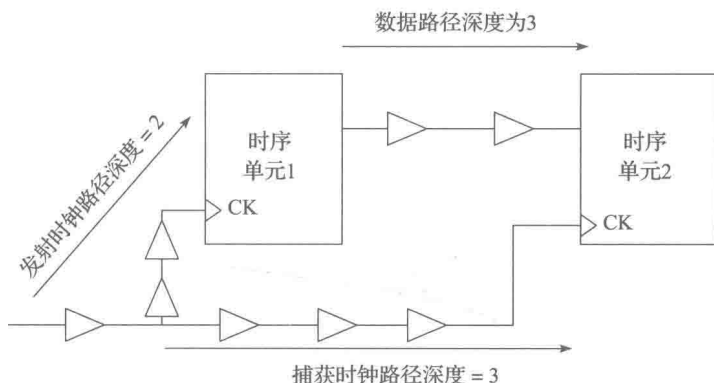


图 5-28 AOCV 分析模式的保持时间计算

如图 5-28 所示,保持时间计算中,根据最快发射时钟路径的深度值 2,通过 AOCV 分析模式对应减免路径类型 (early) 查找表中得到时序减免值为 0.8758,那么发射时钟路径的时序减免值为 0.8758。同理,根据最慢捕获时钟路径对应深度值,通过 AOCV 分析模式对应减免路径类型 (late) 查找表得到对应的减免值为 1.1042,赋值后进行时序分析计算。

2. 参数化芯片变化相关分析模式

前面介绍的 AOCV 分析模式也存在如下缺点:

1) 随着制造工艺越来越先进,比如 14nm 工艺下,AOCV 分析模式中动态设置时序减免值的方法已经不能满足当前时序分析精度的要求,其时序分析结果的悲观性也无法很好地改变。

2) AOCV 分析模式需要先分析时序路径的逻辑深度和物理距离两个因素,再设置时序减免值,导致其时序分析计算工作量增大。因此对于比较大的设计,整个设计的时序分析计算时间会增加。

3) 同样是因为 AOCV 分析模式需要基于分析时序路径的逻辑深度和物理距离因素来设置时序减免值,如果设计中途改动了时序路径中逻辑深度,比如时序优化中增加了缓冲器或者移除了缓冲器,那么其时序路径的逻辑深度改变后就需要重新设置时序减免值,这样会导致时序分析时需要不停地刷新时序减免值设置,并重新计算。

因此,时序分析中引入参数化芯片变化相关分析模式(简称 POCV)来提供比 AOCV 分

析模式更好的时序分析结果悲观性移除效果。

与 AOCV 分析模式动态设置时序减免值的方法不同, POCV 分析模式把单元的最终延时表示为片上随机变化影响的一个计算函数式, POCV 分析模式计算如式 (5-21) 所示。

$$\text{Delay} = \text{Delay}_{\text{nom}} + \text{Delay}_{\text{var}} \cdot P \quad (5-21)$$

其中, $\text{Delay}_{\text{nom}}$ 是指单元正常的平均延时, $\text{Delay}_{\text{var}}$ 是指一倍标准差延时, P 是指标准正态分布 $N(0, 1)$ 。

POCV 分析模式的数据输入格式有两种:

1) 使用与 AOCV 一样的查找表数据输入格式, POCV 分析模式查找表格式如下。

```
version : 4.0
ocvm_type : pocvm
object_type: lib_cell
rf_type : rise fall
delay_type : cell
derate_type: early
object_spec: lib28nm/invx*
coefficient: 0.05
```

如上所示, POCV 分析模式查找表格式中有以下几个重要参数: 版本、对象类型、上拉下拉类型、延时对象类型、减免路径类型、对象属性、片上随机变化系数 (coefficient)。

□ 版本

由于不同版本存在格式、内容和分析模式的区别, 因此版本参数用来定义该 POCV 查找表数据的版本。

□ 对象类型

对象类型参数定义 POCV 用于哪种类型的对象上, 对象分为 3 种: design、lib_cell 和 cell。

□ 上拉下拉类型

由于时序分析主要由单元延时和互连线延时组成, 因此延时类型参数用于定义延时计算时 POCV 对哪种延时类型有效, 延时对象分为 3 种: cell、net 和 cell net。

□ 延时对象类型

由于 POCV 分析中存在最快路径和最慢路径, 因此减免路径类型参数用于定义 POCV 有效时所对应的最快路径或者最慢路径, 减免路径类型分为两种: early 和 late。

□ 减免路径类型

由于 POCV 分析中存在最快路径和最慢路径, 因此减免路径类型参数用于定义 POCV 有效时所对应的最快路径或者最慢路径, 减免路径类型分为两种: early 和 late。

□ 对象属性

根据对象类型参数的定义, 需要定义对应的对象属性参数, 该参数用于定义 POCV 作用于对象的模式。

□ 片上随机变化系数

通过片上随机变化系数可以计算出实际的延时值, 该片上随机变化系数的值可通过

SPICE 蒙特卡洛仿真得到。其系数的计算如式 (5-22) 所示。

$$\text{POCV coefficient} = \frac{\sigma(\text{delay variation})}{\mu(\text{nominal delay})} \quad (5-22)$$

2) 在时序信息文件中通过 POCV 电平转换或负载查找表直接定义。

POCV 电平转换或负载查找表的格式 (liberty variation format) 简称 LVF, 如下所示。

```
ocv_sigma_cell_rise ("delay_template_7x7") {
sigma_type : "early";
index_1("0.0088000, 0.0264000, 0.0608000, 0.1296000, 0.2672000, 0.5424000,
1.0936000");
index_2("0.0010000, 0.0024000, 0.0052000, 0.0108000, 0.0221000, 0.0445000,
0.0895000");
values("0.000476, 0.000677, 0.001075, 0.001870, 0.003438, 0.006626, 0.012922", \
"0.000651, 0.000901, 0.001303, 0.002081, 0.003678, 0.006818, 0.013144", \
"0.000840, 0.001166, 0.001714, 0.002558, 0.004112, 0.007249, 0.013529", \
"0.001115, 0.001520, 0.002193, 0.003317, 0.005087, 0.008153, 0.014445", \
"0.001521, 0.002033, 0.002883, 0.004242, 0.006522, 0.010072, 0.016258", \
"0.002155, 0.002793, 0.003853, 0.005563, 0.008424, 0.012955, 0.020171", \
"0.003204, 0.003977, 0.005321, 0.007515, 0.010960, 0.016582, 0.025786");
}
ocv_sigma_cell_rise ("delay_template_7x7") {
sigma_type : "late";
index_1("0.0088000, 0.0264000, 0.0608000, 0.1296000, 0.2672000, 0.5424000,
1.0936000");
index_2("0.0010000, 0.0024000, 0.0052000, 0.0108000, 0.0221000, 0.0445000,
0.0895000");
values("0.000476, 0.000677, 0.001075, 0.001870, 0.003438, 0.006626, 0.012922", \
"0.000651, 0.000901, 0.001303, 0.002081, 0.003678, 0.006818, 0.013144", \
"0.000840, 0.001166, 0.001714, 0.002558, 0.004112, 0.007249, 0.013529", \
"0.001115, 0.001520, 0.002193, 0.003317, 0.005087, 0.008153, 0.014445", \
"0.001521, 0.002033, 0.002883, 0.004242, 0.006522, 0.010072, 0.016258", \
"0.002155, 0.002793, 0.003853, 0.005563, 0.008424, 0.012955, 0.020171", \
"0.003204, 0.003977, 0.005321, 0.007515, 0.010960, 0.016582, 0.025786");
}
```

5.8 时钟路径悲观移除

静态时序分析工具在计算最快时钟路径与最慢时钟路径延时上可分别选择不同的延时路径或者不同的时序模型文件进行计算, 但是最快时钟路径与最慢时钟路径上可能存在公共时钟路径, 如图 5-29 所示。

那么在时序计算时, 由于公共时钟路径的选择不同会出现不同的公共时钟路径延时值, 而公共时钟路径延时出现差值却不符合实际情况, 从而导致时序结果过于悲观。因此通过设置时钟路径悲观移除 (clock path pessimism removal, CPPR) 来移除时钟公共路径上信号延时计算带来的误差。

这里以 OCV 分析模式下一条典型的时序路径为例, 来介绍 CPPR, 如图 5-30 所示。

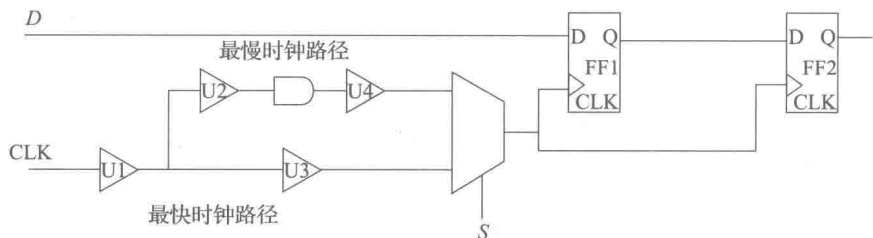


图 5-29 公共时钟路径

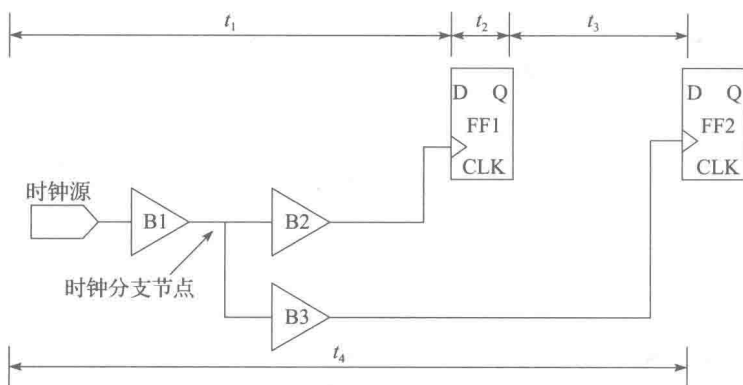


图 5-30 CPPR 介绍

如图 5-30 所示，B1 为发射时钟路径和捕获时钟路径的公共部分。当不考虑 CPPR 时，在 OCV 模式下，建立时间分析中需要满足式 (5-23)。

$$t_{1\max} + t_{2\max} + t_{3\max} \leq t_{4\min} + t_{\text{pa}} - t_{\text{su}} \quad (5-23)$$

其中， $t_{1\max}$ 为发射时钟最慢路径延时值， $t_{2\max}$ 为时序单元 FF1 在最坏情况下的单元延时， $t_{3\max}$ 为数据路径中组合逻辑在最坏情况下的总延时， $t_{2\max} + t_{3\max}$ 为最慢数据路径延时值， $t_{4\min}$ 为捕获时钟最快路径延时， t_{pa} 为时钟周期， t_{su} 为时序单元 FF2 的建立时间要求值。

公共时钟路径 B1 在计算发射时钟最慢路径延时值 t_1 时使用 Max 库，而在计算捕获时钟最快路径延时值 t_4 时使用 Min 库。由于芯片实际工作时 B1 输出点是一个信号驱动后续的发射时钟和捕获时钟信号，而以上公式却把 B1 输出点做为两个不同传播延时的信号进行延时计算，这是不符合实际情况的。

当考虑 CPPR 时，在 OCV 模式下，建立时间分析中需要满足式 (5-24)。

$$t_{1\max} + t_{2\max} + t_{3\max} \leq t_{4\min} + t_{\text{pa}} + t_{\text{su}} + t_{\text{cpr}} \quad (5-24)$$

其中，增加了一个新参数 t_{cpr} ， t_{cpr} 即为时钟公共路径 B1 在发射时钟路径和捕获时钟路径上计算的延时差值。这样计算出来的时序路径延时值就不会过于悲观，更加符合真实情况。

同样，当考虑 CPPR 时，在 OCV 模式下，保持时间分析中需要满足式 (5-25)。

$$t_{1\min} + t_{2\min} + t_{3\min} + t_{\text{cpr}} \leq t_{4\max} + t_{\text{H}} \quad (5-25)$$

其中， $t_{1\min}$ 为发射时钟最快路径延时值， $t_{2\min}$ 为时序单元 FF1 在最好情况下的单元延时， $t_{3\min}$

为数据路径中组合逻辑在最好情况下的总延时, $t_{2\min} + t_{3\min}$ 为最快数据路径延时值, $t_{4\max}$ 为捕获时钟最慢路径延时, t_{cpr} 为时钟公共路径 B1 在发射时钟路径和捕获时钟路径上计算的延时差值, t_{H} 为时序单元 FF2 的保持时间要求值。

5.9 时序优化

解决时序收敛的问题也是静态时序分析中的主要工作。静态时序分析中可以通过自动和手动两种方式完成物理上和时序上的优化工作, 其优化时序的类型主要分为建立时间、保持时间和时序设计规则 3 种。

1. 基本方法

优化建立时间和时序设计规则的基本方法主要为以下几种。

(1) 改变单元位置

不合理的单元物理位置会使信号线过长而导致线负载和线延时变大, 从而影响时序收敛, 如图 5-31 所示。

可以通过优化单元的物理位置来达到提高时序性能的目的, 如图 5-32 所示。

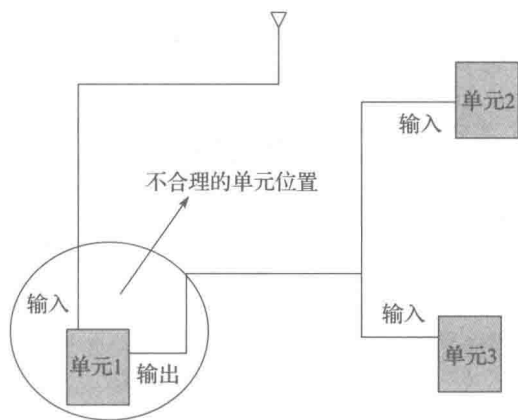


图 5-31 不合理的单元位置

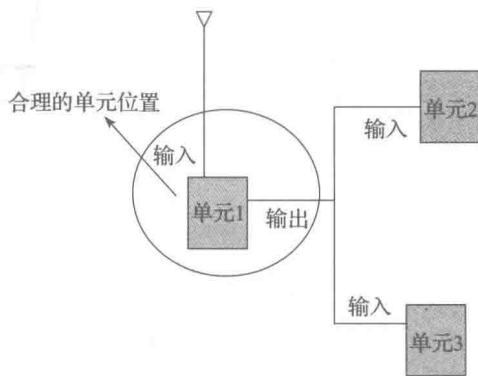


图 5-32 合理的单元位置

(2) 改变单元大小

通常标准单元库提供了多种驱动能力的单元, 由于在相同激励和负载情况下, 其延时与驱动能力近似成线性关系, 延时随驱动能力的增大而递减, 因此把由于负载过大导致延时恶化的单元调整为驱动能力更大的同类型单元, 可以改善时序性能。同样, 驱动能力小的同类型单元其对前级驱动单元所体现的负载电容也较小, 如果将驱动力大的负载单元换成驱动力小的负载单元, 可以减小前级驱动单元的输出负载, 从而改善前级驱动单元的时序性能。

(3) 插入缓冲单元

随着工艺尺寸的不断缩小, 其线延时增加与线长近似成正比指数性关系, 而增加缓冲单

元个数与延时近似成正比线性关系。那么在单元位置比较合理,同时单元驱动力已经达到最大时的情况下,如果单元驱动信号线负载仍然过大而导致延时过大的情况,可以通过在长距离信号线中间增加合适的缓冲单元来提高延时性能,如图 5-33 所示。

(4) 删除缓冲单元

如果设计中存在插入缓冲器单元不合理的情况,那么可以通过删除导致降低时序性能的缓冲单元,来达到优化时序性能的目的。比如在修复建立时间时,由于插入了延时过大的延时单元而导致保持时间违反,而这种延时单元也属于一类特殊的缓冲单元,那么可以删除这种导致降低时序性能的缓冲单元。

(5) 重分配负载

在设计中存在扇出 (fanout) 负载不合理的情况,如图 5-34 所示。

那么可以通过插入缓冲单元重新分配扇出负载来达到优化时序的目的,如图 5-35 所示。

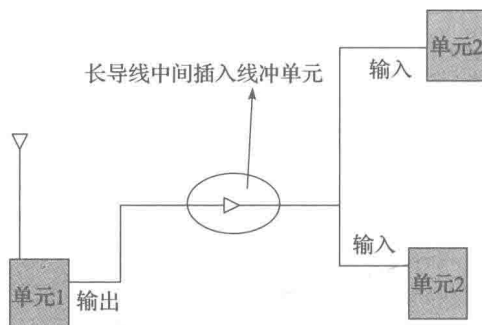


图 5-33 插入缓冲单元

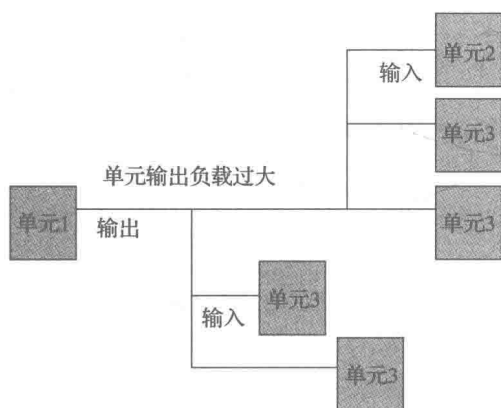


图 5-34 负载不合理

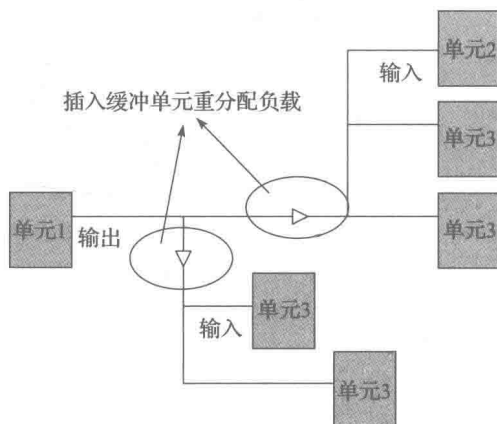


图 5-35 重分配负载

(6) 时钟有用偏斜

时钟有用偏斜 (useful skew) 在前面时钟树性能参数部分已经详细介绍过。它可以用于优化时序性能,其工作原理就是在两个串行的时序路径中,通过借用其中一条时序路径的延时余度的方法来修复另一条时序路径上时序违反的问题。

2. 优化保持时间的基本方法

优化保持时间的方法相对简单,主要是插入延时单元 (delay cell) 的方法。由于延时单元具有相对较长的单元本征延时的特点,通过插入延时单元来显著增加时序路径整体延时来达到满足建立时间要求的目的。

第 6 章

时序约束

静态时序分析工具以时序约束作为判断设计中时序是否满足设计要求的标准，因此设计者需要提供正确的时序约束信息，以便静态时序分析工具输出正确的时序报告。通过 SDC 可以提供简单而又直接的方法来描述未来设计中芯片工作时所预期的时序约束。SDC 的全称为 Synopsys 设计约束，SDC 的格式也是一种业界标准。SDC 一般包括时钟约束、IO 输入输出约束、伪路径约束，多周期约束等内容，这章将对主要的时序约束进行详细说明。

6.1 时钟约束

在同步电路设计中，各功能逻辑单元之间的数据传输由一个同步信号控制，这个执行统一指挥的信号就是时钟信号，因此需要在设计之初创建时钟，基于该时钟频率进行优化设计，使设计性能达到时序收敛的目的。

6.1.1 创建时钟

时钟信号为一个周期性信号，定义时钟需要包括以下主要基本信息：

(1) 时钟源点

时钟源点可以根据情况定义为设计中一个端口，也可以是一个逻辑单元的 PIN 脚。

(2) 时钟周期

时钟周期定义为时钟的振荡周期，为时钟频率的倒数。时钟周期是时序分析中最基本的、最小的时间单位。

(3) 时钟占空比

时钟占空比主要定义时钟高低电平在一个时钟周期内的分布情况。

通过使用 SDC 命令 `create_clock` 来创建时钟（`create clock`），比如通过该命令定义一个从 CLK 的端口输出并且时钟周期为 10、占空比为 50% 的时钟信号，如图 6-1 所示。

示例命令如下。

```
create_clock -period 10 -waveform {0 5} [get_ports clk]
```

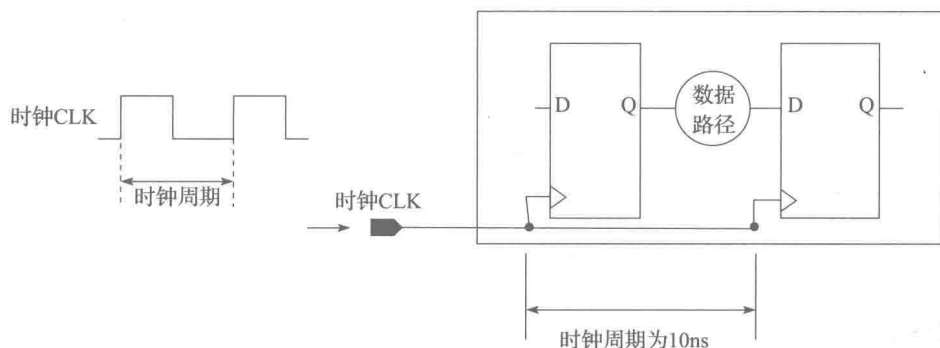


图 6-1 创建时钟信号

通过该命令，可以对所有的内部触发器到触发器类型的路径进行约束。

占空比为 50% 的设置是默认的，即如果没有定义占空比信息，则默认占空比为 50%。

下面定义一个占空比不为 50% 的时钟，示例命令如下：

```
create_clock -name BDYCLK -period 15 -waveform {5 12} [get_ports CLK]
```

时钟信号波形如图 6-2 所示。

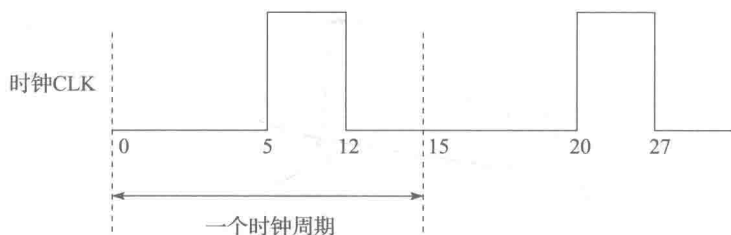


图 6-2 时钟信号波形

(4) 时钟转换延时

时钟转换延时定义时钟在高低电平状态下切换所需要的延时。

通过使用 SDC 命令 `set_clock_transition` 来定义时钟转换延时信息，如图 6-3 所示。

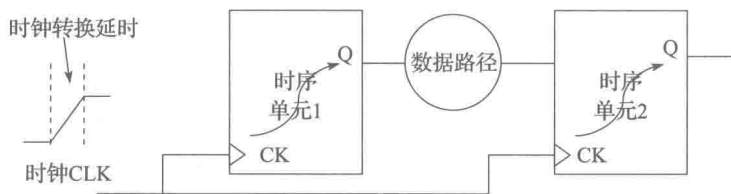


图 6-3 时钟转换延时

示例命令如下：

```
set_clock_transition -rise 0.1 [get_clocks clk]
```

```
set_clock_transition -fall 0.12 [get_clocks clk]
```

以上命令定义名为 clk 的时钟信号上升转换延时为 0.1ns，下降转换延时为 0.12ns。

(5) 时钟不确定性

由于实际时钟本身与理想时钟有一定的误差，所以通过时钟不确定来涵盖这些实际的导致误差的因素，比如时钟抖动、时钟偏斜等因素。

由于芯片内部各个触发器的位置不同，导致每个触发器到时钟源点的距离不相同，所以时钟源点的时钟信号到达每个触发器的时间与理想情况的时间有误差。如图 6-4 所示，时钟源点到达触发器 P4 点的时间为 1.3ns，而到达触发器 P1 点的时间为 0.5ns，这样，时钟不确定性的延时为 0.8ns。

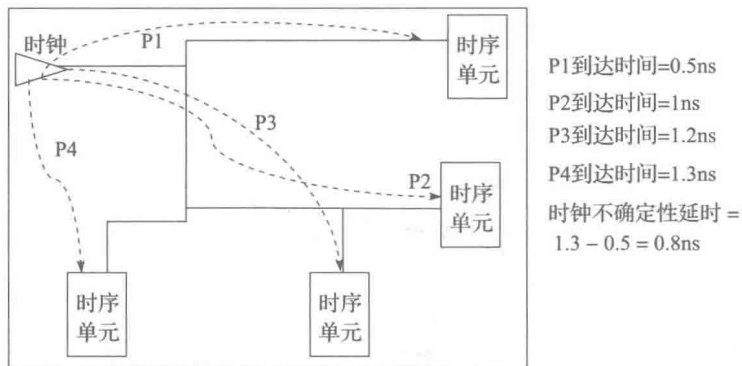


图 6-4 时钟不确定性

通过使用 SDC 命令 `set_clock_uncertainty` 来定义时钟不确定性信息，示例命令如下：

```
set_clock_uncertainty -setup 0.2 [get_clocks clk]
set_clock_uncertainty -hold 0.05 [get_clocks clk]
```

以上命令定义名为 CLK 时钟信号的不确定性为：建立时间为 0.2ns，保持时间为 0.05ns。其结果如图 6-5 所示。

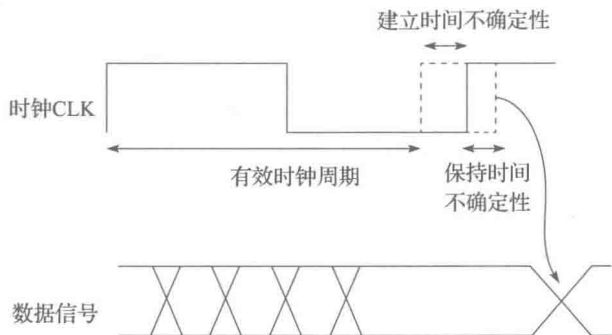


图 6-5 不确定性定义结果

设计中可能存在由不同的时钟信号驱动的时序路径，如图 6-6 所示。

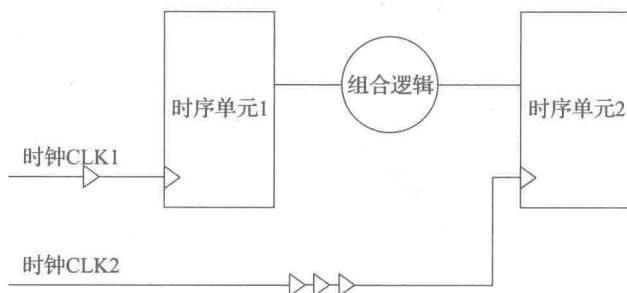


图 6-6 不同时钟的时序路径

不同时钟域之间也可以定义时钟不确定性，示例命令如下：

```
set_clock_uncertainty -from CLK1 -to CLK2 -hold 0.05
set_clock_uncertainty -from CLK2 -to CLK1 -hold 0.05
set_clock_uncertainty -from CLK1 -to CLK2 -setup 0.1
set_clock_uncertainty -from CLK2 -to CLK1 -setup 0.1
```

以上命令定义名为 CLK1 和 CLK2 时钟之间的不确定性为：建立时间为 0.1ns，保持时间为 0.05ns。

（6）时钟延迟

时钟信号从时钟源输出端到达时序单元时钟输入端是需要传播时间的。时钟延迟由时钟源点到达时钟定义的端口的延时（source latency）和时钟定义的端口到时序单元时钟输入端的延时（network latency）两部分组成，如图 6-7 所示。

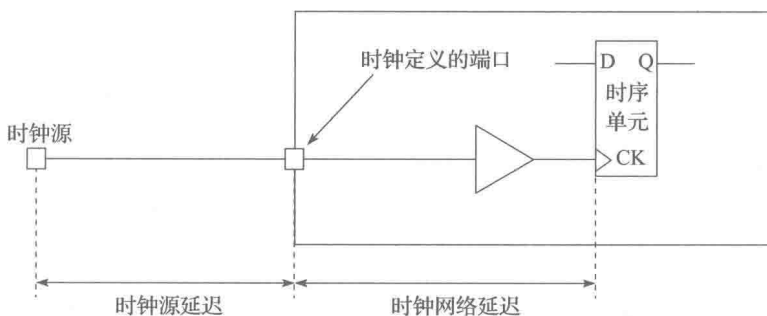


图 6-7 时钟延迟

通过使用 SDC 命令 `set_clock_latency` 来定义时钟延迟信息，示例命令如下：

```
set_clock_latency 1.8 -rise [all_clocks]
set_clock_latency 2.1 -fall [all_clocks]
```

以上命令定义了时钟网络延迟：上升延迟为 1.8ns，下降延迟为 2.1ns。

```
set_clock_latency 0.851 -source -min [get_clocks CLK]
set_clock_latency 1.322 -source -max [get_clocks CLK]
```

以上命令定义了时钟源信号到达时钟 CLK 输出端的延时，最大延迟为 1.322ns，最小延迟为 0.851ns。

时钟网络延迟与时钟源延迟的一个明显区别是，时钟网络延迟是时钟树生成前的设置。当实际时钟树生成后，时钟网络延迟的设置将会通过命令 `set_propagated_clock` 被时钟树的实际结果替代，而时钟源延迟会一直存在，即使使用实际的时钟树传播延时信息。

6.1.2 生成时钟

在设计中有这样一类时钟，它是基于一个主时钟并通过相关逻辑转换后，在相位、频率、占空比等方面和主时钟有一定变化的分支时钟。我们可以把这类时钟定义为生成时钟，因此，生成时钟的时钟源来自主时钟，其相位参考主时钟，如图 6-8 所示。

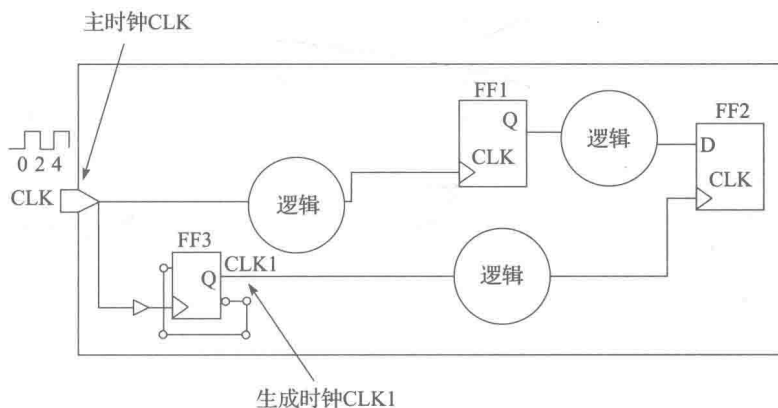


图 6-8 生成时钟

通过使用 SDC 命令 `create_generated_clock` 来定义生成时钟（generated clock），示例命令如下：

```
create_clock -period 10 -waveform {0 5} [get_ports CLK]
create_generated_clock -name CLKDIV3 -source CLK -divide_by 3 [get_pins DFF/Q]
```

以上命令定义一个基于主时钟 CLK 的分支时钟 CLKDIV3，CLKDIV3 的频率是主时钟的 1/3，即生成时钟的周期是主时钟的 3 倍。生成时钟 CLKDIV3 与主时钟 CLK 的时钟信号波形的对比图如图 6-9 所示。

可以对生成时钟在相位上进行设置，示例命令如下：

```
create_clock -period 10 [get_ports CLK]
create_generated_clock -name CLKDIV2 -divide_by 2 -invert -source CLK [get_pins DFF/Q]
```

以上命令定义一个基于主时钟 CLK 的分支时钟 CLKDIV2，CLKDIV2 的周期为主时钟的 2 倍，其相位与主时钟相反。生成时钟 CLKDIV2 与主时钟 CLK 的时钟信号波形的对比图如图 6-10 所示。

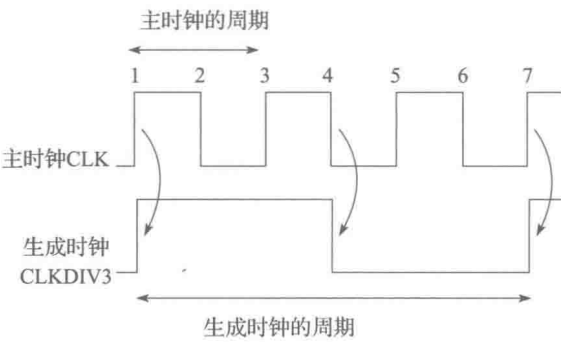


图 6-9 时钟信号波形对比图 1

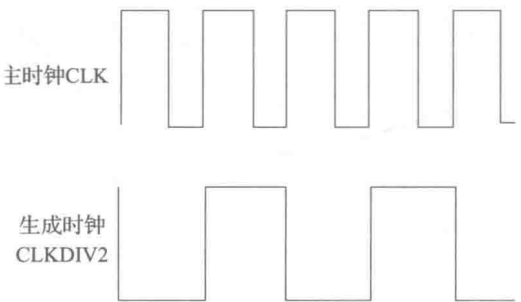


图 6-10 时钟信号波形对比图 2

可以对生成时钟在占空比上进行设置，示例命令如下：

```
create_clock -period 10 -waveform {0 5} [get_ports CLK]
create_generated_clock -name CLK1 -source CLK -edges {1 1 5} -edge_shift {0 5 0}
[get_pins DFF/Q]
```

以上命令定义一个基于主时钟 CLK 的分支时钟 CLK1，CLK1 的周期为主时钟的 2 倍，其占空比为高电平 25%。生成时钟 CLK1 与主时钟 CLK 的时钟信号波形的对比图如图 6-11 所示。

生成时钟在实现上也可以通过创建时钟命令来设置，而区别是创建时钟命令是得到一个独立的时钟信号，这样增加一个独立时钟域就会相应增加静态时序分析的工作量。同时创建时钟还需要设置时钟相关的其他基本信息，比如时钟源、时钟延迟，时钟不确定性等，而生成时钟的优点则是会自动继承主时钟的设置，无需额外设置。如图 6-12 所示为生成时钟继承主时钟的时钟延迟的情况。

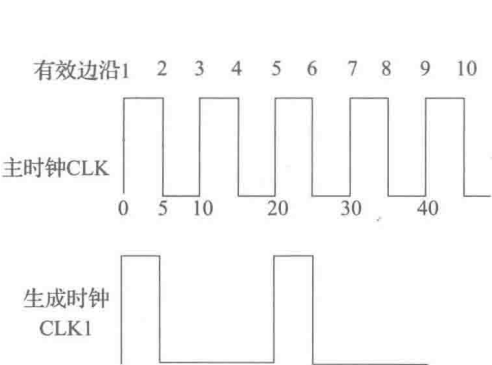


图 6-11 时钟信号波形对比图 3

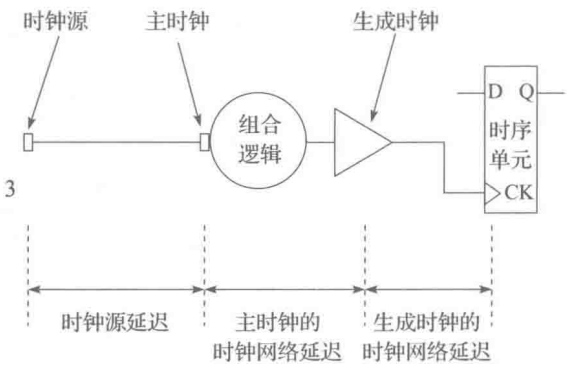


图 6-12 生成时钟继承主时钟的时钟延迟

生成时钟继承主时钟的设置，因此生成时钟的时钟延迟如下。

$$\begin{aligned} \text{生成时钟的时钟延迟} &= \text{时钟源延迟} + \text{主时钟的时钟网络延迟} \\ &+ \text{生成时钟的时钟网络延迟} \end{aligned}$$

生成时钟也存在缺点，由于生成时钟只能对应一个主时钟，因此在某些具体应用中使用创建时钟命令来实现时钟比较合适。比如某个时钟是通过多个时钟信号混合后转换而来，在各个时钟带来的不确定性影响共同作用下，其实际时钟的输出结果，比如相位和占空比等属性与任何一个时钟没有绝对对应关系时，那么通过把其中任何一个时钟作为主时钟来定义生成时钟时就可能出现误差。对于这种情况，把最终时钟输出端单独使用创建时钟来定义比较安全。

如图 6-13 所示为两个时钟信号通过与门转换后得到最终的时钟输出的情况。

如果把 CLK3 定义为生成时钟，在 CLK1 和 CLK2 各自带来的时钟抖动、时钟延迟等因素共同作用下，CLK3 的输出结果并不完全取决于其中某一个时钟，即 CLK3 的输出结果实际应该继承两个时钟的共同结果，所以使用其中任何一个时钟作为主时钟都不合理。这样直接设置 CLK3 为独立的时钟，并通过设置其他时钟相关的基本信息参数来涵盖需要考虑的不确定因素是比较合理的做法。

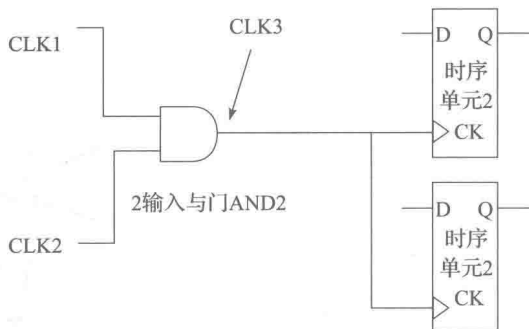


图 6-13 两个时钟信号相与的时钟输出

由于一个主时钟可以对应多个生成时钟，所以不同的生成时钟可以来自同一个主时钟，这在实际设计中也是很正常的应用。比如芯片中不同的时钟域都来自同一个时钟源点：锁相环（PLL）部件的输出，基本时钟网络结构如图 6-14 所示。

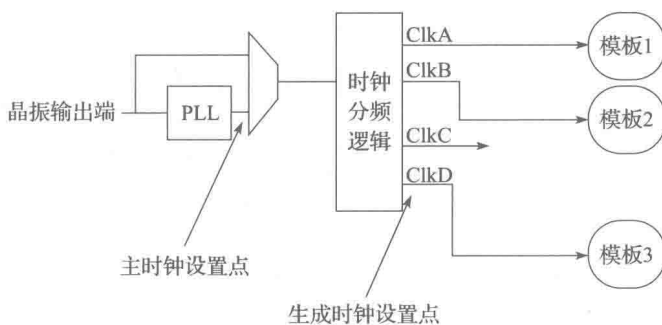


图 6-14 基本时钟网络结构

如图 6-14 所示，生成时钟 ClkA、ClkB、ClkC 和 ClkD 都是通过分频逻辑来自同一个主时钟信号：锁相环的输出。

根据以上情况,设计本身的时钟为 CLK1,但是其输入端口 core_in 是来自外部基于 CLK2 时钟域下的时序路径的输出,其示例命令如下。

```
create_clock -name CLK2 -period 10 -waveform {0 5}
create_clock -name CLK1 -period 10 [get_ports core_ck]
set_input_delay -clock CLK2 -max 2.7 [get_ports core_in]
```

以上命令定义设计本身的时钟 CLK1 和定义虚拟时钟 CLK2,同时基于虚拟时钟来定义输入端口 core_in 的时序约束,其输入延时为 2.7ns。

□ 设计的输出端口信号进入外部某个时钟控制下的时序路径输入端,如图 6-17 所示。

根据以上情况,设计本身的时钟为 CLK1,但是其输出端口 core_out 是到自外部基于 CLK2 时钟域下的时序路径的输入,其示例命令如下。

```
create_clock -name CLK2 -period 10 -waveform {0 5}
create_clock -name CLK1 -period 10 [get_ports core_ck]
set_output_delay -clock CLK2 -max 2.7 [get_ports core_out]
```

以上命令定义设计本身的时钟 CLK1 和定义虚拟时钟 CLK2,同时基于虚拟时钟来定义输出端口 core_out 的时序约束,其输出延时为 2.7ns。

□ 设计的输入、输出端口信号都是外部某个时钟控制下的时序路径,如图 6-18 所示。

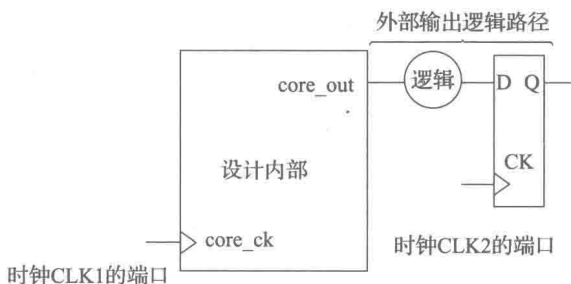


图 6-17 外部时钟控制的输入端

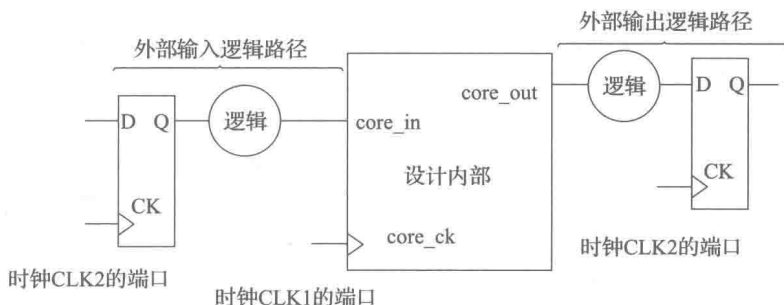


图 6-18 外部时钟控制的输入输出端

根据以上情况,设计本身的时钟为 CLK1,但是其输入、输出端口是到自外部基于 CLK2 时钟域下的时序路径,其示例命令如下。

```
create_clock -name CLK2 -period 10 -waveform {0 5}
create_clock -name CLK1 -period 10 [get_ports core_ck]
set_input_delay -clock CLK2 -max 2.7 [get_ports core_in]
```

```
set_output_delay -clock CLK2 -max 2.7 [get_ports core_out]
```

以上命令定义设计本身的时钟 CLK1 和定义虚拟时钟 CLK2，同时基于虚拟时钟来定义输入、输出端口的时序约束，其输入、输出延时都定义为 2.7ns。

6.1.4 最小时钟脉宽

时钟信号的脉宽如果太小会引起如下两个方面的问题：

1) 时序单元无法正常工作。

由于时序单元本身有建立时间和保持时间的约束，需要时钟信号脉宽必须保持一段时间的稳定，如图 6-19 所示。

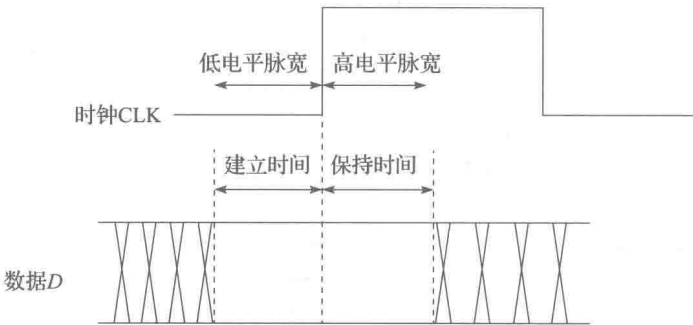


图 6-19 时钟信号脉宽需求 1

从图 6-19 可以看出，时钟信号高低电平的最小脉宽至少分别要大于等于满足建立时间与保持时间所需要的时间，否则时序单元无法正常工作。

2) 任何信号都不能脉宽太小，否则在组合逻辑路径传播过程中会慢慢削弱而失真，如图 6-20 所示。

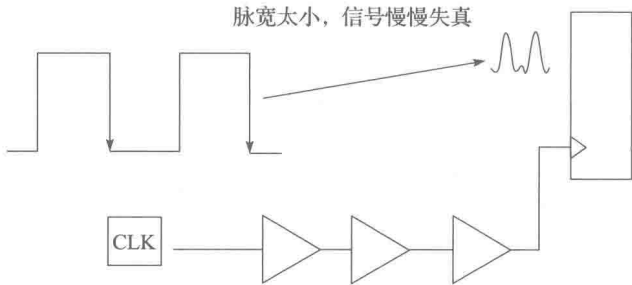


图 6-20 时钟信号脉宽需求 2

同样时钟信号也会因为脉宽太小的问题在时钟树中无法正确地传播，时钟信号的失真可能会导致整个设计功能出现错误。

由于以上两个原因，需要设置时钟信号脉宽保持稳定的最短时间。

通过使用 SDC 命令 `set_min_pulse_width` 来定义时钟信号的最小脉宽 (minimum pulse width), 示例命令如下:

```
set_min_pulse_width -high 1.5 [all_clocks]
set_min_pulse_width -low 1.0 [all_clocks]
```

以上命令定义所有时钟高电平最小脉宽为 1.5ns, 低电平最小脉宽为 1ns。

6.2 I/O 延时约束

前面介绍了 4 种时序路径中的 3 种: 触发器到输出端、输入端到触发器和输入端到输出端, 都与外部 IO 相关, 所以在时序分析中需要对外部 IO 的延时进行时序约束, 如图 6-21 所示。

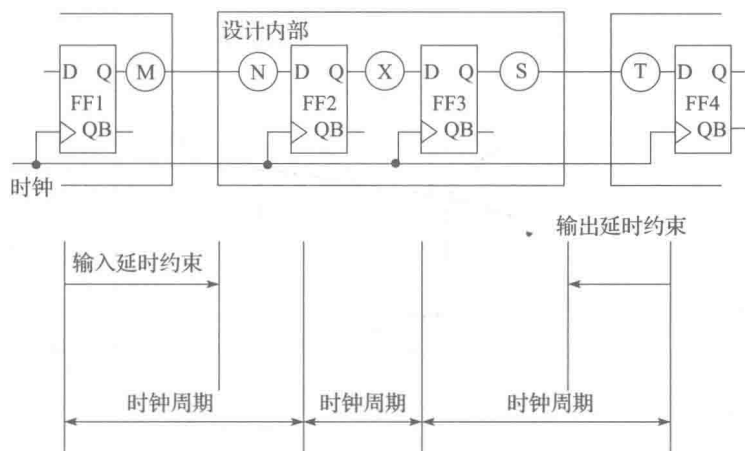


图 6-21 外部 IO 延时约束

这样可以在对这 3 种类型的时序路径进行时序分析时考虑到外部 IO 延时的信息。

1. 设置输入延时

输入延时 (input delay) 定义为在一个有效时钟周期内, 外部逻辑的输出数据到达设计输入端口所占用的延时, 如图 6-22 所示。

通过使用 SDC 命令 `set_input_delay` 来定义 IO 输入延时, 示例命令如下:

```
set_input_delay -min 2.0 -clock CLK [get_ports IN]
set_input_delay -max 2.0 -clock CLK [get_ports IN]
```

以上命令定义一个基于时钟 CLK 的外部逻辑到输入端口 IN 的延时信息, 其延时波形如图 6-23 所示。

2. 设置输出延时

输出延时 (output delay) 定义为在一个有效时钟周期内, 输出端口数据到外部逻辑所占用的延时, 如图 6-24 所示。

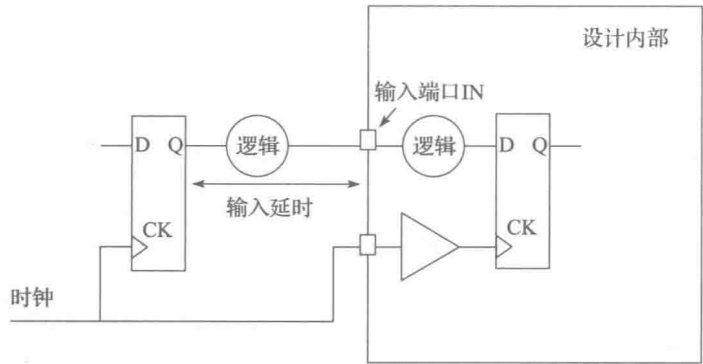


图 6-22 输入延时

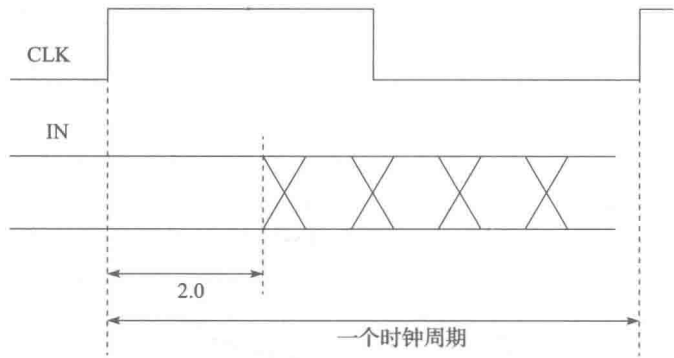


图 6-23 输入延时波形

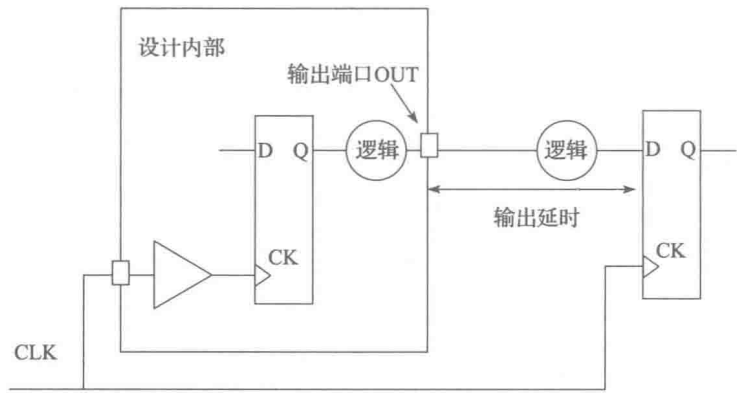


图 6-24 输出延时

通过使用 SDC 命令 `set_output_delay` 来定义 IO 输入延时，示例命令如下：

```
set_output_delay -clock CLK -max 1.5 [get_ports OUT]
set_output_delay -clock CLK -min 1.5 [get_ports OUT]
```

以上命令定义一个基于时钟 CLK 的输出端口 OUT 到外部逻辑的延时信息，其延时波形如图 6-25 所示。

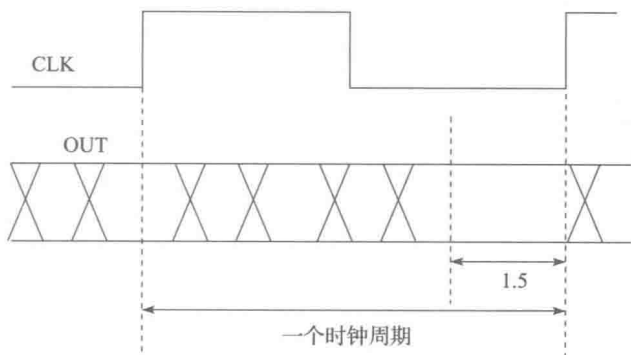


图 6-25 输出延时波形

6.3 I/O 环境建模约束

前面介绍的 I/O 延时约束只能对 I/O 的输入、输出延时信息进行设置，但是对于输入、输出外部的实际负载情况并没有做精确的设置，如图 6-26 所示。

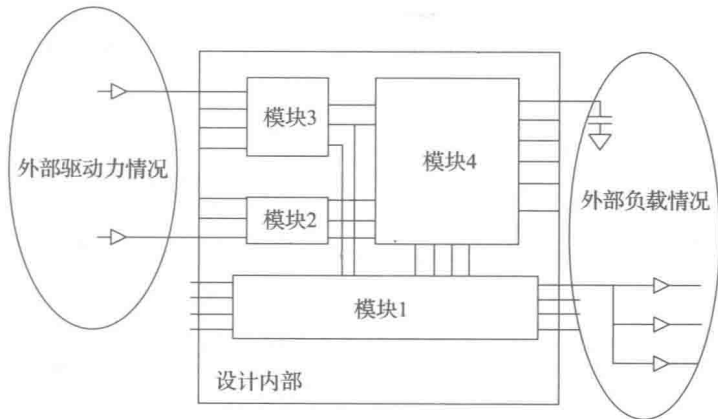


图 6-26 输入输出外部情况

如果缺少比较真实的外部输入、输出信息，时序分析工具对与外界逻辑有连接关系的内部逻辑的输入、输出延时的计算结果就会和实际情况之间有比较大的误差。可以通过设置 I/O 环境建模约束来设置与实际情况相符合的外部输入驱动力信息和输出负载信息，这样可以指导时序分析工具计算出处在内部与外部交界上的单元的更符合实际情况的延时信息。

I/O 环境建模约束的内容主要包括以下几个方面。

1. 输入驱动建模

默认情况下, 如果没有外部输入驱动力信息的设置, 时序分析工具默认设置外部输入驱动力为无限大。输入驱动建模的实现方法可以通过以下 3 个命令实现。

1) 通过使用 SDC 命令 `set_drive` 来设置外部输入驱动力信息, 通过该命令定义驱动该引脚的输入驱动电阻值, 再通过该驱动电阻值算出驱动端的转换时间, 并计入时序路径的延时信息里。设置的输入驱动电阻值越大, 其输入端驱动力越小, 当定义为 0 时, 代表输入端驱动力为无限大, 如图 6-27 所示。

示例命令如下:

```
set_drive -rise 0.4 [all_inputs]
set_drive -fall 0.3 [all_inputs]
```

以上命令定义设计所有输入的驱动值, 上拉驱动电阻为 0.4, 下拉驱动电阻为 0.3。

2) 通过使用 SDC 命令 `set_driving_cell` 来设置外部输入驱动力信息, 通过该命令定义驱动该引脚的输入驱动力等价为一个指定单元的驱动能力, 再通过该等价单元的驱动力算出驱动端的转换时间, 并计入时序路径的延时信息里。设置的等价单元驱动能力大则其输入端驱动能力大, 如图 6-28 所示。

示例命令如下:

```
set_driving_cell -lib_cell buff2 [all_inputs]
```

以上命令定义设计所有输入的驱动力等价于单元库里 buff2 单元的驱动力。

3) 通过使用 SDC 命令 `set_input_transition` 来设置外部输入驱动力信息, 通过该命令直接定义驱动该引脚的输入驱动端转换时间。设置的转换时间越大其驱动力越小, 当定义为 0 时, 代表输入端驱动力为无限大, 如图 6-29 所示。



图 6-27 设置外部输入驱动力方法 1

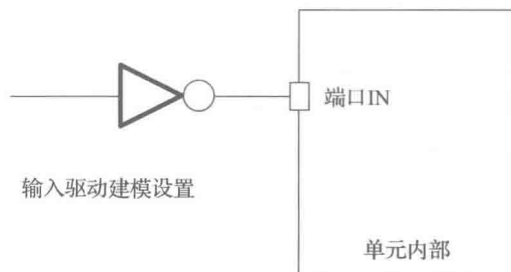


图 6-28 设置外部输入驱动力方法 2

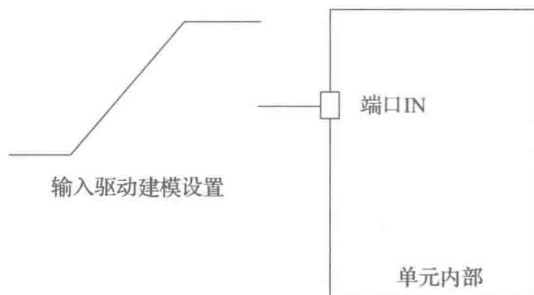


图 6-29 设置外部输入驱动力方法 3

示例命令如下：

```
set_input_transition 0.6 [all_inputs]
```

以上命令定义设计所有输入的驱动力的转换时间为 0.6ns。

2. 输出负载建模

默认情况下，如果没有外部输出负载信息的设置，时序分析工具默认设置外部输出负载为 0。输出负载建模的实现方法可以通过使用 SDC 命令 `set_load` 实现，通过该命令定义外部负载为一个等效的电容，其电容值即为负载值，如图 6-30 所示。

示例命令如下：

```
set_load 5 [all_outputs]
```

以上命令定义设计所有输出负载等效为 5pF 的电容负载。

输出负载建模也可以通过该命令定义输出负载等效为某指定单元引脚的电容值，如图 6-31 所示。

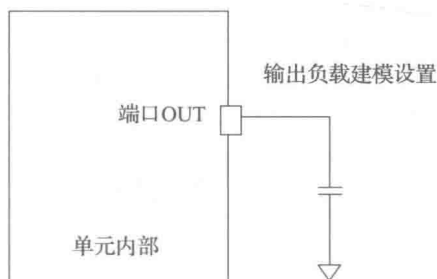


图 6-30 设置外部输出负载方法 1

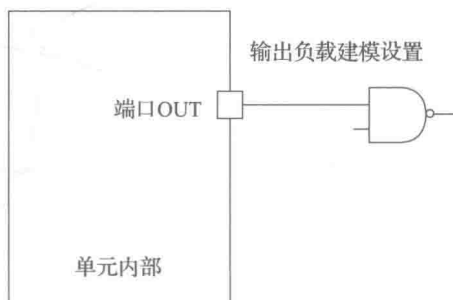


图 6-31 设置外部输出负载方法 2

示例命令如下：

```
set_load [get_attribute [get_lib_pins tech_lib/NAND2/A] pin_capacitance] [all_outputs]
```

以上命令定义设计所有输出负载等效为与非门 NAND2 的引脚 A 的电容负载。

6.4 时序例外

静态时序分析工具默认分析规则是在单周期下对所有时序路径进行分析，但是实际设计中并不是所有的路径都需要分析，而且也不是所有时序路径的延时都规定在一个时钟周期以内。所以通过定义时序例外来对特殊的时序路径进行特殊的声明，指导时序分析工具对特殊的时序路径做出正确分析，这样的特殊声明叫做时序例外（timing exception）。

时序例外主要用于如下几种情况。

1. 多周期路径设置 (multicycle path)

设计中存在时序路径延时比较长, 并且设计允许信号在这类时序路径下的延时等于几个周期的时间, 如图 6-32 所示。

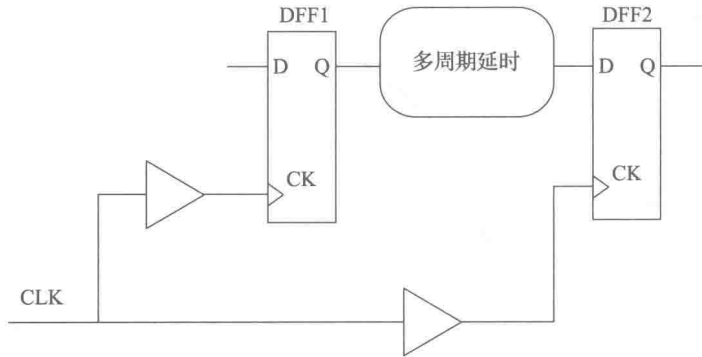


图 6-32 多周期路径

那么可以通过使用 SDC 命令 `set_multicycle_path` 来定义这类时序路径为多周期时序路径。示例命令如下:

```
set_multicycle_path 5 -setup -from [get_pins DFF1/Q] -to [get_pins DFF2/D]
set_multicycle_path 4 -hold -from [get_pins DFF1/Q] -to [get_pins DFF2/D]
```

以上命令定义触发器 DFF1 与 DFF2 之间时序路径建立时间的延时为第 5 个时钟周期检查, 保持时间的延时为第 4 个周期检查, 其延时波形如图 6-33 所示。

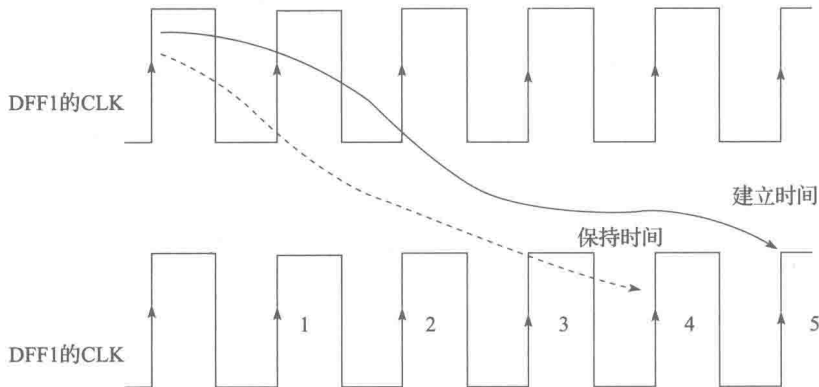


图 6-33 多周期路径延时波形

2. 伪路径设置 (false path)

当设计中存在不需要做时序分析的时序路径时, 可以通过伪路径的设置移除该时序路径上的默认时序约束, 指导时序分析工具对设置的时序路径停止时序分析。

一般设计中满足如下情况的时序路径可以定义为伪路径:

1) 设计中存在某些时序路径在某特定的工作状态下是不工作的, 那么在对应的工作状态下进行时序分析时, 该时序路径实际上是不需要进行分析的。

比如测试逻辑的时序路径, 这种路径在正常工作模式下是不需要考虑的, 这种时序路径在工作模式下的时序分析中就可以定义为伪路径, 如图 6-34 所示。

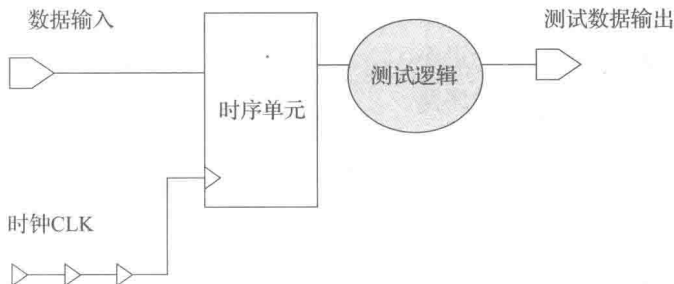


图 6-34 伪路径类型 1

2) 设计中存在某些时序路径的时序要求是非常的宽松, 以至于不需要进行时序分析。

如, 复位控制信号的时序路径, 如图 6-35 所示。

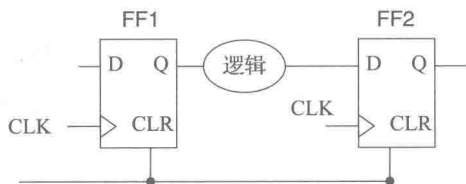
这种路径在设计中一般对时序要求比较宽松, 所以根据设计要求可以定义为伪路径。

再如, 具有反馈环的时序路径如图 6-36 所示。

这种路径一般在设计中如果不设置伪路径, 可能出现时序违反, 但是这种违反是可以接受的, 那么可以定义为伪路径。

3) 设计中存在某些时序路径并不会真正的工作信号通过, 这种时序路径实际上是无效的时序路径。

如, 不同时钟域之间的时序路径, 如图 6-37 所示。



复位信号 (RESET)

图 6-35 伪路径类型 2

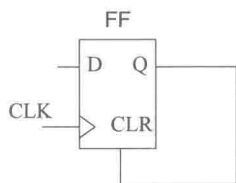


图 6-36 具有反馈环的时序路径

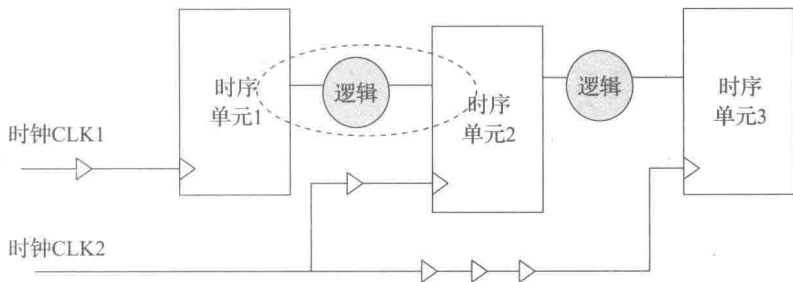


图 6-37 伪路径类型 3

这种时序路径在设计中可能存在无关性，那么可以定义为伪路径。

再如，设计中数据信号通过某些逻辑单元的部分时序路径没有对应输入激励来触发工作，如图 6-38 所示。

两个同样的选择器由于选择信号不一样，因此其中的部分时序路径一定不会被触发，那么可以把不工作的时序路径定义为伪路径。

伪路径通过使用 SDC 命令 `set_false_path` 来定义，示例命令如下：

```
set_false_path -from [get_clocks CLK1] -to [get_clocks CLK2]
```

以上命令定义两个时钟域 CLK1 和 CLK2 之间的时序路径为伪路径。

```
set_false_path -through [get_pins DFF/RST]
```

以上命令定义通过单元 DFF 中 RST 引脚的时序路径为伪路径。

3. 最大延时和最小延时设置

当设计需要对某些点到点路径进行特殊设置时，可以通过定义最大延时 (Maximum Delay) 和最小延时 (Minimum Delay) 来实现。比如，时钟域 clk1 和 clk2 之间的时序路径默认设置是伪路径，但是其中有一条从触发器 F1 通过组合逻辑 1 到触发器 F2 的时序路径需要有时序约束，如图 6-39 所示。

那么可以对这条路径通过定义最大延时和最小延时来实现，该命令将会覆盖掉所设置路径的默认时序约束。

通过使用 SDC 命令 `set_min_delay` 和 `set_max_delay` 来定义最大延时和最小延时，示例命令如下。

```
set_max_delay 0.6 -from DFF1/Q -to DFF2/D
set_min_delay 0.2 -from DFF1/Q -to DFF2/D
```

以上命令定义单元 DFF1 到单元 DFF2 之间时序路径的时序约束的最大延时为 0.6ns，最小延时为 0.2ns。

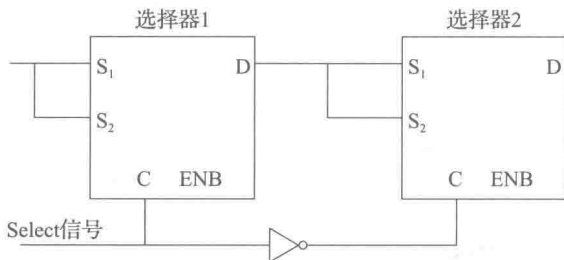


图 6-38 和输入无关的路径

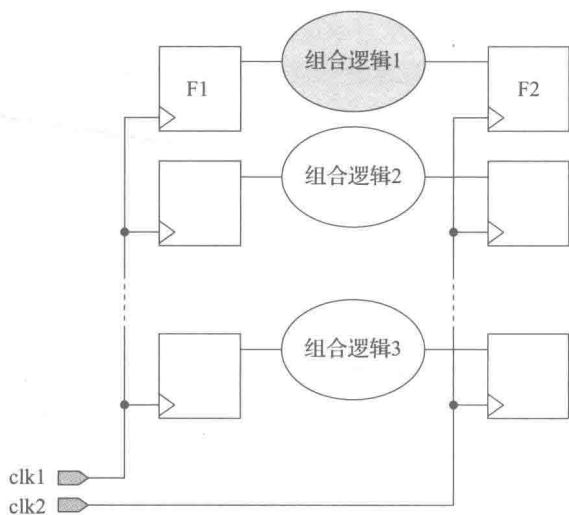


图 6-39 点到点路径特殊设置

当最大延时和最小延时设置命令中约束的节点定义的是时序路径中的节点时，那么时序约束仅仅应用于两个节点之间。而当约束的节点定义的是一个时钟，最大延时和最小延时设置将应用于该时钟源所控制的所有路径，示例命令如下。

```
set_max_delay 1.2 -from [get_clocks CLK1] -to [get_clocks CLK2]
set_min_delay 0.4 -from [get_clocks CLK1] -to [get_clocks CLK2]
```

以上命令定义从时钟域 CLK1 到 CLK2 之间所有时序路径的时序约束的最大延时为 1.2ns，最小延时为 0.4ns。

6.5 恒定状态约束

根据实际的设计需求，在某种工作模式下有些信号实际上为恒定的值，比如自测试逻辑中的 TEST 引脚在测试模式下应该设置为 1，在常规工作模式下就应该一直保持为 0，如图 6-40 所示。

如果没有正确的恒定状态约束，测试逻辑的时序延时是很难满足常规工作模式下的时序要求的，因此需要通过做相应的恒定状态约束来指导时序分析工具进行正确的时序分析。

通过使用 SDC 命令 `set_case_analysis` 来定义恒定状态约束，示例命令如下：

```
set_case_analysis 0 [get_port test_scan_enable]
```

以上命令定义测试模式端口 `test_scan_enable` 恒定状态值为 0。

再比如，设计工作在不同的设计频率下也可以通过恒定状态约束来实现，如图 6-41 所示。

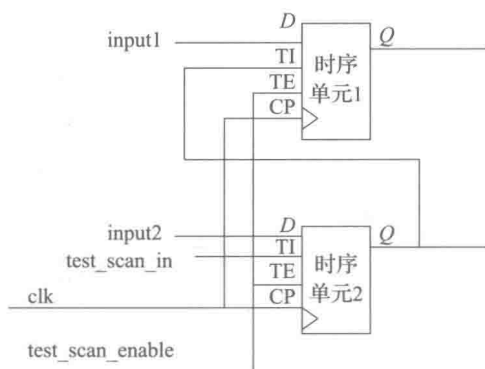


图 6-40 恒定状态约束情况 1

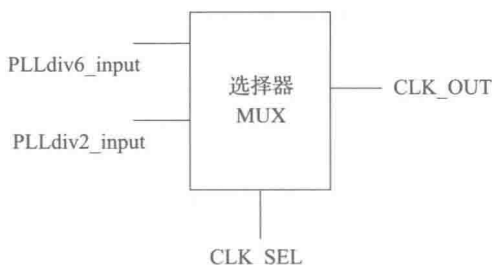


图 6-41 恒定状态约束情况 2

根据设计的工作模式需要，通过设置恒定状态约束来定义时钟选择端口的对应状态值，选择对应的时钟频率进行正确的时序分析。示例命令如下。

```
set_case_analysis 1 CLK_SEL
```

以上命令定义测试模式端口 `CLK_SEL` 的恒定状态值为 1。

6.6 屏蔽时序弧

前面提到过，每个单元内部从输入端口到输出端口都定义有时序弧，默认情况下所有的时序弧都应该有效并用于时序分析。但是在设计中由于具体逻辑设计的原因，某些单元内部的时序弧由于没有对应功能逻辑的触发条件而应该是无效的，对于这样的特殊情况可以通过屏蔽时序弧的约束命令来实现，以指导时序分析工具进行正确的时序分析。

比如，选择器单元存在选择输入端到输出端的时序弧，但是设计中不存在通过该单元选择端口到输出端口的时序路径，这样可以通过屏蔽该时序弧，指导时序分析跳过包含该时序弧的时序分析，如图 6-42 所示。

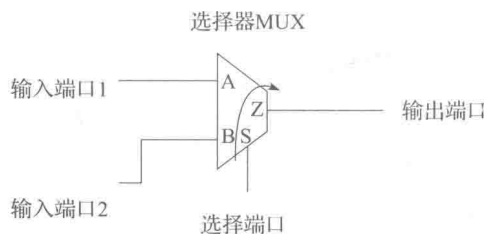


图 6-42 屏蔽时序弧情况

屏蔽时序弧设置通过 SDC 命令 `set_disable_timing` 来定义。示例命令如下：

```
set_disable_timing -from S -to Z [get_cells MUX]
```

以上命令定义屏蔽选择器单元 MUX 中从 S 端到 Z 端的时序弧。

屏蔽时序弧 (`set_disable_timing`) 和伪路径 (`set_false_path`) 在某种情况下存在相似的实现结果，但是也存在本质的不同。屏蔽时序弧是针对单元具体端口内部的时序弧并且进行屏蔽，等价于移除掉该单元内部的时序弧。那么时序分析时，与该时序弧相关的所有时序路径都被移除，如图 6-43 所示。

而伪路径是针对时序路径并移除对应时序路径上的默认时序约束，而所有时序弧是一直有效的，如果与单元同一时序弧相关的时序路径有多条，而伪路径设置只作用于其中部分时序路径，那么时序工具默认还是会计算所有时序路径延时，只是对存在时序约束的时序路径进行时序性能评估，如图 6-44 所示。

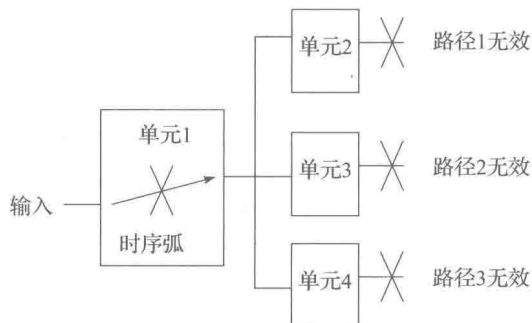


图 6-43 屏蔽时序弧结果

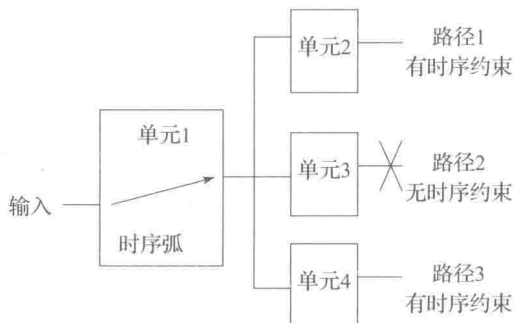


图 6-44 伪路径时序弧结果

当伪路径设置是移除通过某单元具体 PIN 脚上的所有时序约束时，命令如下所示：

```
set_false_path -through [get_pins ADDER/CI]
```

可以用屏蔽时序弧约束替代，命令如下所示：

```
set_disable_timing [get_pins ADDER/CI]
```

这样的好处是屏蔽时序弧命令可以减少时序分析工具的计算负载，因为对应时序弧已经无效，不会进行计算分析。

6.7 时序设计规则约束

在设计中，由于时序路径中的逻辑单元本身存在驱动能力的限制，所以在满足时序要求的同时也需要保证逻辑单元本身工作在稳定的合理工作状态下，即逻辑单元所驱动的负载应该控制在可以接受的范围内，如图 6-45 所示。

因此可以通过设置设计规则约束来指导时序分析工具正确地评估单元是否能够工作在稳定的合理工作状态下。

时序设计规则约束的内容主要包括以下 3 个方面。

1. 最大转换时间

由于所有的单元其驱动能力是有限的，所以其驱动的最大转换时间（max transition）也是有限的。为了让设计中的所有单元都在正常的驱动力范围内工作，需要设置最大的转换时间（max transition）来控制单元的驱动力过载的问题，否则可能会导致不能正常工作。

通过使用 SDC 命令 `set_max_transition` 来定义最大转换时间约束，示例命令如下：

```
set_max_transition 0.6 [all_outputs]
```

以上命令定义所有输出驱动端的最大转换时间为 0.6ns。

2. 最大电容负载

前面讲到的单元驱动转换时间本质上取决于输出负载的大小，因此单元的驱动状态根本原因是受到输出端的负载影响。为了让设计中的所有单元都在正常的驱动力范围内工作，也可以设置单元输出端的最大电容负载（max capacitance），超过最大负载值的限制有可能会導致单元不能正常工作。

通过使用 SDC 命令 `set_max_capacitance` 来定义最大电容负载约束，示例命令如下：

```
set_max_capacitance 0.5 [all_outputs]
```

以上命令定义所有输出驱动端的最大电容负载为 0.5pF。

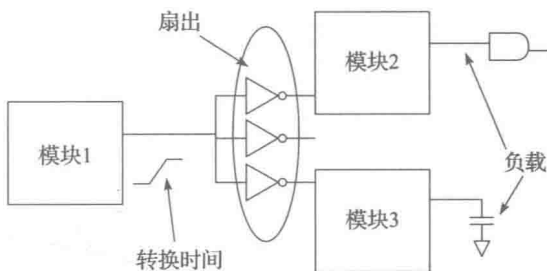


图 6-45 时序设计规则约束

3. 最大扇出

扇出是指逻辑单元输出端直接连接的下级逻辑单元输入端的个数，由于单元输出端所连接的输入端的个数多少直接影响负载的大小，所以为了让设计中的所有单元都在正常的驱动力范围内工作，也可以设置单元输出端的最大扇出值（max fanout），超过最大扇出值的限制有可能会使单元不能正常工作。

通过使用 SDC 命令 `set_max_fanout` 来定义最大扇出约束，示例命令如下：

```
set_max_fanout 16 [all_outputs]
```

以上命令定义所有输出驱动端的最大扇出值为 16。

由于以上 3 种时序设计规则约束是在设计中根据实际情况人为设置的约束，一般其值应该略低于实际单元库真正承受的值，这样可以为设计留有一定的裕度，实际的极限值以单元库属性文件（lib 库）中设置的值为准。

第 7 章

串 扰 噪 声

集成电路进入超深亚微米设计后，串扰噪声就成为影响芯片功能和性能的重要原因之一，因此，合理解决串扰噪声带来的不利影响是当前集成电路设计中必须面对的越来越重要的问题。

7.1 噪声的定义

在 VLSI 设计中，噪声是指电路中任意节点上不希望出现的电压或电流的偏差。当噪声作用于某个电路节点上时，它可以瞬间破坏该节点上储存的逻辑信息。如果这一破坏状态被锁存到锁存器中，那么将会导致功能性错误。即使噪声没有导致功能性错误，它也会影响电路的时序（延时和斜率）。而且，噪声引起的不必要的逻辑翻转也会导致功耗的增加。

在数字电路中，与噪声相关的两个重要概念如下。

1. 噪声容限

一个逻辑门对噪声的灵敏度是由噪声容限 NM_L （低电平噪声容限）和 NM_H （高电平噪声容限）来度量的，它们分别量化了合法的“0”和“1”的范围，并确定了噪声的最大固定阈值，如式（7-1）和式（7-2）所示：

$$NM_L = V_{IL} - V_{OL} \quad (7-1)$$

$$NM_H = V_{OH} - V_{IH} \quad (7-2)$$

噪声容限表示当逻辑门互连在一起时所能允许的噪声电平，如图 7-1 所示。

为了能够使一个数字电路工作正常，这一容限应当大于 0，并且越大越好。噪声容限用于衡量一个电路相对于噪声的稳定性标准。

2. 抗噪声能力

抗噪声能力是指逻辑门在噪声存在的情况下正确处理和传递信息的能力。有许多数字电路，它们的噪声容限很小，但却有很好的抗噪声能力，这是因为它们能抑

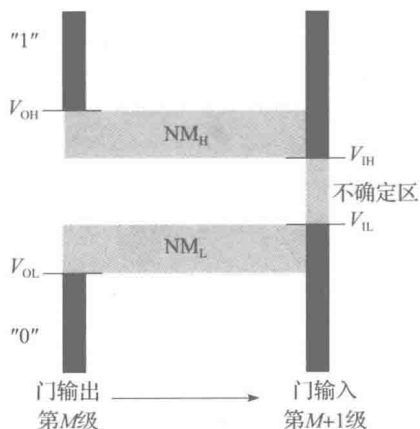


图 7-1 噪声容限

制噪声源。如差分逻辑，它们能抑制大多数的内部噪声，因而在噪声容限和信号摆幅很小时也不会受影响。

7.2 噪声的来源

在深亚微米工艺下将遇到很多复杂的电学问题，这些问题包括线间串扰引起的信号扭曲、电源分布引起的电压降、衬底耦合、电荷共享、电荷泄露、工艺偏差、热噪声和阿尔法粒子等。下面将针对深亚微米工艺下对电路影响最大的噪声进行详细的分析。

1. 电容耦合

电容耦合噪声是指两条相邻信号线之间由于耦合电容的存在，当一条信号线进行翻转时（干扰线）就会在另一条信号线上（受扰线）注入一定的电荷，从而引起受扰线上的电压发生偏差。这种方式引起的噪声通常具有脉冲的形态，即它的跳变沿由干扰线的翻转斜率决定，而它的恢复沿由受扰线的恢复时间常数决定。这是在当前工艺下最主要的噪声来源。

电容耦合噪声主要分为两种类型：功能性噪声和延时噪声。当有噪声注入受扰线上时，就会在受扰线上产生一个毛刺（glitch），如图 7-2 所示。

如果这个毛刺没有及时恢复，而传播到下游的锁存单元，被锁存单元捕获，导致锁存错误，那么就会产生一个功能性噪声，引起功能性错误。并不是每个毛刺都会造成功能性错误，只有那些幅度超出了电路的噪声容限并且不能及时恢复的毛刺，才有可能产生功能性噪声。

第二种串扰噪声类型是延时噪声。当受扰线与干扰线同时翻转时，受扰线的翻转时间就会发生改变，这种瞬态延时的改变会影响线上的传播延时和瞬态转换时间，从而可能造成建立时间或保持时间不满足时序约束，如图 7-3 所示。

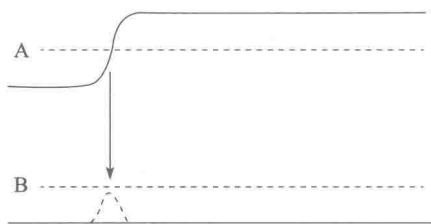


图 7-2 耦合产生的毛刺

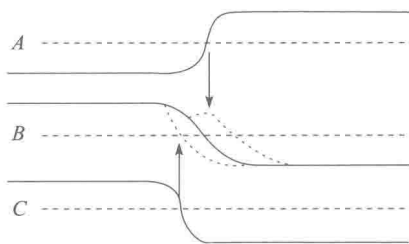


图 7-3 噪声引起的翻转提前或滞后

如图 7-3 所示， B 是受扰线， A 和 C 是干扰线，在电容耦合的作用下， A 和 C 的翻转会影响 B 的瞬态转换时间。 A 从“0”到“1”的翻转会导致 B 的瞬态转换时间变慢，可能导致 B 的建立时间违反；反之， C 从“1”到“0”的翻转使得 B 的瞬态转换时间变快，可能导致 B 的保持时间违反。

串扰的程度由以下 5 个因素决定：

- 相邻线之间耦合电容的大小
- 受扰线和干扰线的驱动能力
- 翻转的相对时间间隔
- 线上的总电容
- 线上的总电阻

2. 电荷分享

电荷分享是指动态电路中动态评估节点和上拉或下拉网络中的内部节点之间的电荷重分配现象,如图 7-4 所示。

这种由于电荷分享导致的噪声主要出现在动态多米诺逻辑、传输管逻辑和动态锁存器等电路中。

3. 电荷泄漏

电荷泄漏主要是指一个晶体管在正常关断的情况下还存在亚阈漏流,这个电流通过对动态节点进行充电或者放电,导致动态节点的逻辑电平偏离原来的轨道。这种噪声主要发生在大扇入的多米诺 NOR、PLA 和存储网络中。由于亚阈漏流随着阈值电压的下降而成指数上升,所以在 DSM (有限状态机) 设计中,电荷泄漏是一个很重要的问题。一般可以通过添加反馈管来克服这种问题,如图 7-5 所示。

4. IR 压降

IR 压降 (drop) 问题是指出现在半导体电路中电源和地网络上电压下降或升高的一种现象。随着半导体工艺的发展,金属互连线的宽度越来越窄,导致它的电阻值上升,所以在整个芯片范围内存在一定的 IR 压降。IR 压降的大小决定于从电源引脚到逻辑门单元间的等效电阻的大小,如图 7-6 所示。

如图 7-6 所示,当晶体管有开关动作时,假设逻辑单元 P2 的电源电压为 V_{DD} , P2 所消耗的电流为 I_2 ,而其他逻辑单元的电流都为 0,电流 I_2 通过电源网络从外部电源流向 P2,那么逻辑单元 P2 所处的 V_{DD} 上的 IR 压降如式 (7-3) 所示。

$$IR\ drop_{P2} = I_2 \times (R_1 + R_2) \quad (7-3)$$

逻辑单元 P1 的 V_{DD} 上的 IR 压降如式 (7-4) 所示。

$$IR\ drop_{P1} = I_2 \times R_1 \quad (7-4)$$

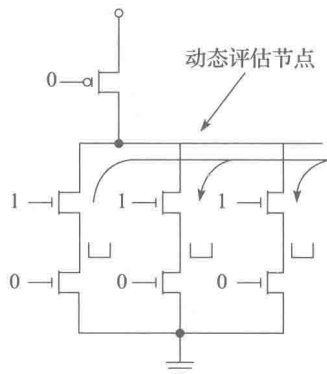


图 7-4 电荷分享噪声

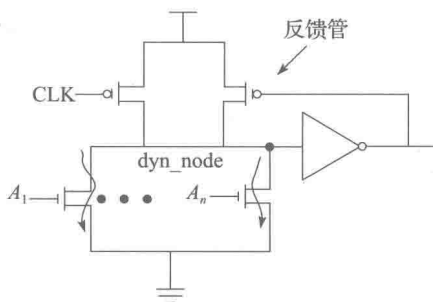


图 7-5 电荷泄漏

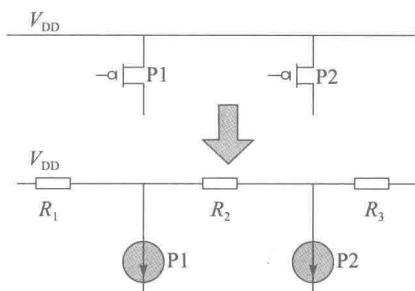


图 7-6 等效电路

因此，芯片设计中的每一个逻辑单元的电流都会对设计中的其他逻辑单元造成不同程度的 IR 压降。如果连接到金属连线上的逻辑单元同时有翻转动作，那么因此而导致的 IR 压降将会很大。然而，设计中某些部分的同时翻转又是非常重要的，例如时钟网络和它所驱动的寄存器，在一个同步设计中它们必须同时翻转。因此，一定程度的 IR 压降是不可避免的。

IR 压降可能是局部的或全局的。当相邻位置一定数量的逻辑门单元同时有逻辑翻转动作时，就会引起局部 IR 压降现象；而电源网络某一特定部分的电阻值特别高时，也会导致局部 IR 压降；当芯片某一区域内的逻辑动作导致其他区域出现 IR 压降时，称之为全局现象。

IR 压降问题的表现常常类似一些时序甚至可能是信号的完整性问题。如果芯片的全局 IR 压降过高，则可能出现功能错误，使芯片彻底失效。局部 IR 压降只在一些特定的条件下才可能发生，例如总线上的信号同时翻转，这时芯片会间歇性地表现出一些功能故障。

典型的设计准则是将 IR 压降限制在电源电压的 10% 以内，但这样的电源电压下降也将使单元延时增大 10%。不同单元的 IR 压降不同，不仅使各基本逻辑门单元的工作电压不同，而且导致下一个连接的逻辑门单元的栅电压减小，电源线的噪声容限降低。IR 压降效应会增大时钟网络的斜率，从而增加数据的保持时间要求；或者增大信号的斜率，增加信号的建立时间要求。这些都会造成数据信号的传输延迟和电平不可预测，导致数据传输错误。随着工艺发展，元件几何尺寸变小，电源和地导线的阻抗相对增加，IR 压降效应对芯片的时序和可靠性影响会更大。Cadence 的白皮书中指出，IR 从 1.7V 降到 1.6V 会导致延时增加 50%，甚至更多。在采用 $0.18\mu\text{m}$ 以及更先进工艺的设计失败案例中，有 20% 是由于压降效应造成的。

由于 IR 压降对延时、功能等方面的影响，所以有必要采取措施减小设计中的 IR 压降。首先可以通过减小电源网络上的电阻来实现，但这种方法受到芯片面积和布线空间的限制；其次可以在电源线上加退耦电容和采用异步电路设计方案；最后，采用铜等低 K 值的金属作为芯片互连线也可以降低导线电阻，从而减小 IR 压降效应。

5. 互感效应

信号翻转时，在信号线和其电流回路形成的环路中产生一个瞬态电流，从而产生一个变化的电磁场，这就是互感效应如图 7-7 所示。

如图 7-7 所示，互感效应产生的这个电磁场会在环路或者环路周围的信号线上产生一个电压。如果在总线上有几根信号线同时翻转，那么这些噪声就会累积起来。

互感是一种大范围内的现象，在同步高速切换的总线结构中噪声将变得非常严重。如果电感和电容噪声叠加起来将会使得问题更加严重，分析更加困难，因为电感分析高度依赖版图的布局并且非常复杂。解决问题的方法通常是根据一定的规则去设计电路，而不是分析任意一种组合并有针对性地去逐个解决。

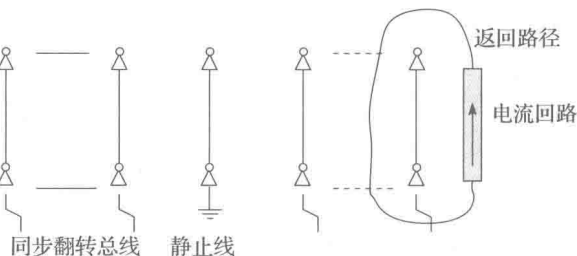


图 7-7 同步翻转导致的互感效应

6. 热效应

热效应是指电源线和信号线上高速翻转的信号产生的电流引起的线自热现象。热效应应在以下3个方面影响电路的性能和可靠性:

1) 为了控制温度的增加, 必须限制信号线上允许的最大电流密度, 因为电流密度太大会导致热效应明显。

2) 导线的寿命(取决于电迁移 EM 速度)与线上的温度成指数下降。由于 EM 能力与电流密度成正比, 所以线自热现象也限制了电路中最大允许的平均电流密度。

3) 热效应有可能造成电路开路问题和潜在的电迁移问题。随着工艺进步, 电流密度不断增加, 热噪声持续增加, 成为了每个电路设计者必须关心和解决的问题。

7. 工艺偏差

在现实中, 一个晶体管的参数对于不同的晶圆会有所不同, 甚至在同一个芯片上的晶体管之间也会由于位置的不同而有差别。在电路相同的器件间出现随机分布的工艺偏差主要是由以下两个原因造成的:

1) 由于积淀或杂质扩散期间的不均匀情况引起工艺参数不同。这些导致了薄层电阻以及如阈值电压这样的晶体管参数值的差异。

2) 器件尺寸上的变化, 主要来自光刻过程有限的分辨率。

7.3 噪声恶化的原因

随着超深亚微米新工艺技术的不断进步, 由于串扰噪声带来的不良影响越来越严重, 其原因有如下几个。

1. 布线密度加大

由于新工艺的特征尺寸、线宽和线间距都将进一步缩小, 新工艺在保持面积不变的情况下, 能把更多的器件整合在片上系统中, 这无疑将大大增加布线的复杂度, 使得线路更加拥挤, 相互之间存在的交叉耦合电容越来越大, 因此带来的相互影响也越来越明显。

2. 金属线厚度增加导致侧面积增加

由于金属阻抗与其宽度成反比, 为追求更小的阻抗, 新工艺下金属线的厚度不断增加, 甚至大于宽度, 且金属线的长度比以往更长, 这种趋势也导致金属之间的交叉耦合电容增加, 因此带来的相互影响也越来越明显, 如图 7-8 所示。

3. 布线层增多

新工艺下, 由于芯片上叠加了更多的金属层, 高金属层与基底层之间的距离加大, 从而减小了对地的电容量, 使金属线之间交叉耦合电容的比重不断上升, 因此串扰效应给设计带来的影响也越来越大。

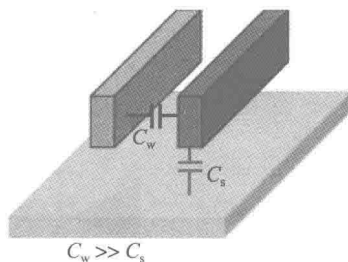


图 7-8 新工艺下的金属线

4. 工作频率增高

随着设计要求越来越高，其芯片的工作频率越来越高，高频工作下的器件充放电时所产生的噪声源也越来越多，加上耦合电容越来越大，因此由于串扰叠加带来的影响也越来越严重。

5. 工作电压降低

在新工艺下，由于电源电压越来越低，其噪声容限也越来越低，导致串扰效应给设计带来的影响也越来越大。

7.4 噪声的体现形式

串扰噪声的分析主要是分析受扰线与干扰线之间的关系，如图 7-9 所示。

并根据如下几点因素来分析干扰线与受扰线之间的相互影响。

(1) 干扰线与受扰线之间的耦合电容

干扰线与受扰线之间的耦合电容越大，其相互影响越大。

(2) 干扰线的驱动信号切换强度大小和切换速度

干扰信号的驱动信号强度越大其高低电平切换时速度也越快，切换速度越快越容易通过耦合电容注入更多的噪声从而影响受扰线。

(3) 受扰线的对地电容

受扰线的对地电容越小，抗干扰能力越弱。

(4) 受扰线的驱动信号强度大小。

受扰线的驱动信号强度越小，抗干扰能力越弱。

前面介绍过，根据串扰噪声对电路的危害的体现形式不同可将其分为两类。

1) 功能噪声 (functional noise)，如图 7-10 所示。

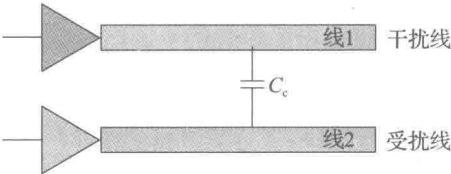


图 7-9 受扰线与干扰线

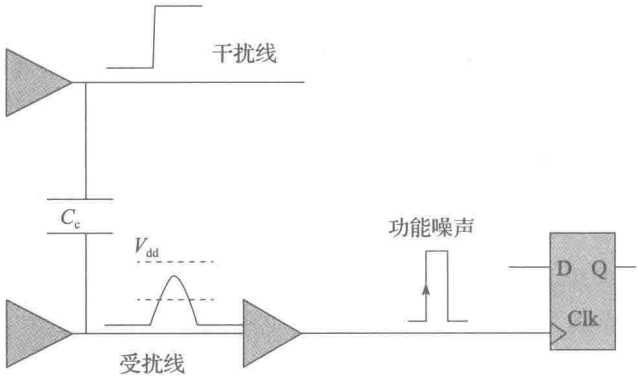


图 7-10 功能噪声

如图 7-10 所示,这种类型的串扰噪声是指受扰线原本处于静止状态,由于干扰线电压的变化而在其上引起毛刺,当毛刺超过噪声容限且改变了原有信号值时(这取决于受扰线原始状态值和干扰线电压变化方向),如果恰被存储部件捕捉,电路状态就会发生改变,导致功能性错误。

2) 延迟噪声(delay noise),如图 7-11 所示。

如图 7-11 所示,这种类型的串扰噪声是指受扰线本身即处于电压变化过程中,此时干扰线的电压变化有可能加快或减缓其变化(这取决于受扰线与干扰线电压变化同向还是反向),引起时序的不确定。如果受扰线电压变化减慢且处于关键路径,可能导致建立时间无法满足设计需要,影响电路工作频率;而如果受扰线电压变化增快,又容易引起保持时间违背(hold time violation),同样会使电路工作失常。

7.5 噪声相互作用形式

串扰问题伴随着半导体工艺集成度的提高,电压的不断降低,越来越成为无法忽视的因素。尤其是流片工艺进入 $0.09\mu\text{m}$ 以下,互连线代替逻辑门成为集成电路的主要时延,这就使得过去还处于次要地位的串扰问题显得更加突出,成为集成电路性能进一步提高的主要障碍。串扰噪声根据互连线之间的关系可以分为干扰线与受扰线两种,干扰线与受扰线输入信号相互关系分为如下 4 种情况。

1) 在受扰线电压变化时,干扰线电压保持不变。

为了分析方便和直观,近似认为干扰线上各个点的电压均保持不变,那么对于受扰线可以近似认为引入了一个等效的接地电容,如图 7-12 所示。这一效应使得受扰线上延迟增大。

2) 受扰线没有电压变化,干扰线上有电压变化。

由于存在耦合电容,受扰线上会出现一个小的干扰脉冲信号,如图 7-13 所示。

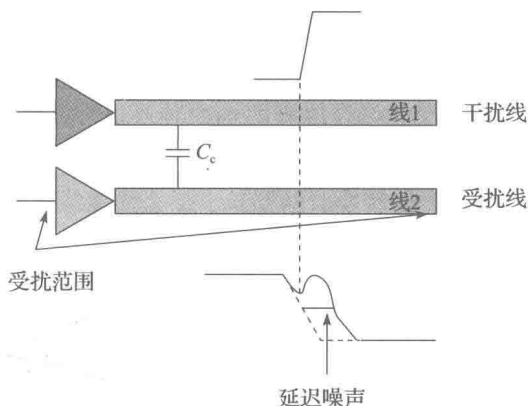


图 7-11 延迟噪声

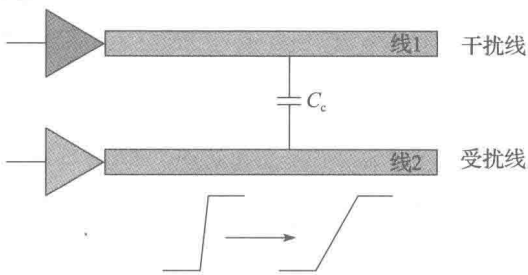


图 7-12 干扰关系 1

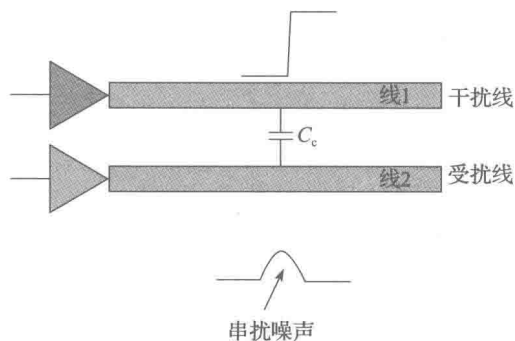


图 7-13 干扰关系 2

对于局部互连线来说，由于耦合比较弱，这个干扰脉冲的峰值比较小，不会引起电路的逻辑错误。而少数全局互连线可能与多条互连线间都有很强的耦合，在某些情况下，干扰脉冲会导致电路出现逻辑错误。

3) 干扰线和受扰线电压同时反方向变化。

在反向变化情况下，串扰会导致受扰线上的延迟变化，如图 7-14 所示。

根据密勒效应，耦合电容等效为一个大小可变的接地电容，加大了干扰线的不确定负载。耦合电容的大小取决于电容开关因子，其值受耦合电容两端电压影响，最大可以有 2 倍大小的变化，但是只有在电容两端的电压波形完全对称时，耦合电容大小才增大 2 倍。

4) 干扰线和受扰线电压同时同方向变化。

仅仅在耦合电容两端的转换信号波形相同时，一般可以忽略耦合电容的影响，因此减少了总电容大小。但是实际情况不可能做到转换信号波形完全相同，如果受扰线的转换时间比干扰线的转换时间要长，那么受扰线在转换过程中也会受到串扰影响而加速转换时间，如图 7-15 所示。

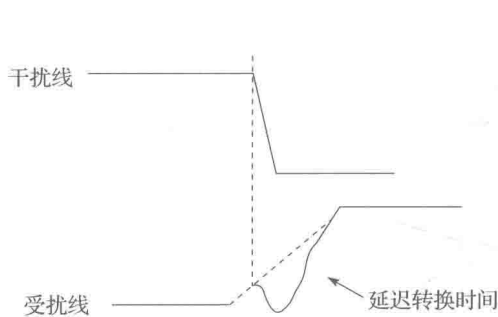


图 7-14 干扰关系 3

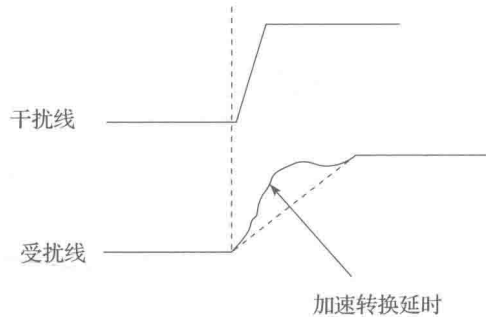


图 7-15 干扰关系 4

以上 4 种情况只是对串扰的简单直观的理想分析，实际电路中影响串扰的因素除了信号间相互关系外，还有信号的转换速度、信号之间相位偏移等因素。

7.6 NLDM 噪声模型的计算

时序分析中的信号完整性是指信号能够抵抗邻近信号的高频电磁干扰效应并且在电路中产生正确响应的能力，如图 7-16 所示。

在高速深亚微米设计中，保持信号完整性对于芯片设计的成功与否至关重要。

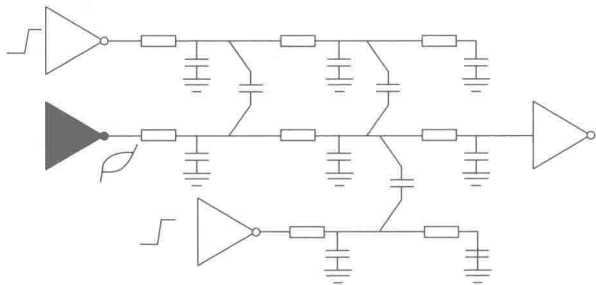


图 7-16 信号完整性相关的高频电磁干扰效应示意

噪声信息建模需要考虑的问题如下：

- 1) 需要了解单元的驱动能力同时计算注入的毛刺。
- 2) 需要计算毛刺的影响，包括直流与交流情况下的噪声容限。
- 3) 需要了解毛刺在路径传播中是否衰减或增强。

所以噪声模型由驱动电阻、噪声容忍度、噪声传播 3 个部分组成。

驱动电阻的计算方法一般是通过 SPICE 仿真 I - V 特性曲线来确定，并且为了覆盖大部分实际情况下的驱动电阻状态，仿真的电压区间一般为 $-V_{DD} \sim 2V_{DD}$ ，如图 7-17 所示。

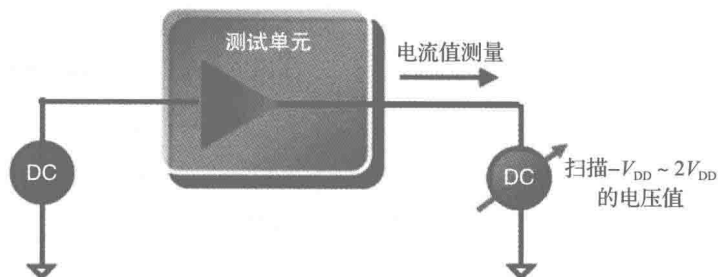


图 7-17 驱动电阻的计算方法

驱动电阻的仿真 I - V 特性曲线如图 7-18 所示。

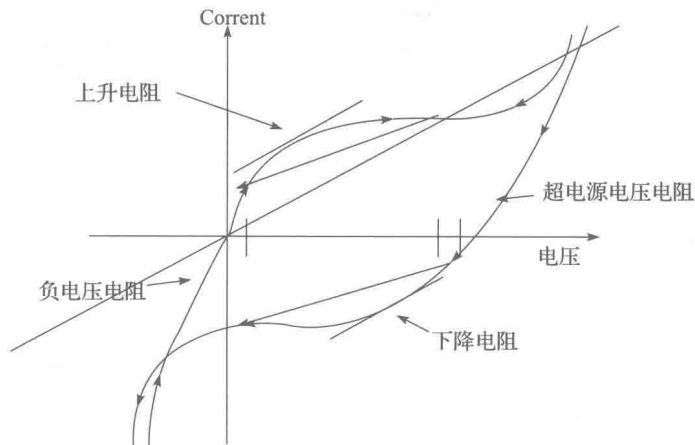


图 7-18 驱动电阻 I - V 曲线图

驱动电阻的仿真结果以电流值的形式保存在时序信息文件的一维查找表中，在时序信息文件中通过 `steady_state_current_low` 和 `steady_state_current_high` 这两个组来定义。驱动电阻的电流值查找表如图 7-19 所示。

噪声容忍度以毛刺能量的形式表示，用于计算逻辑门单元的输入端不同毛刺大小对逻辑门的影响情况。毛刺能量影响示意图如图 7-20 所示。


```
lv_lut_template(template_namestring) {  
    variable_1: lv_output_voltages;  
    index_1 ("floating_number,..., floating_number");  
}  
  
steady_state_current_low(iv_ov_7) {  
    index_1 ("-1.2, -0.6, 0, 0.6, 1.2, 1.8, 2.4");  
    values ("-0.1, -0.05, 0.01, 0.1, 0.25, 0.75, 1.3");  
}  
  
steady_state_current_high(iv_ov_7) {  
    index_1 ("-1.2, -0.6, 0, 0.6, 1.2, 1.8, 2.4");  
    values ("-0.8, -0.1, 0.5, 1, 1.4, 1.7, 1.8");  
}  
}
```

图 7-19 驱动电阻的电流值查找表

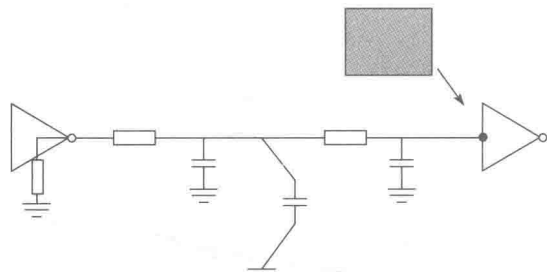


图 7-20 毛刺能量影响示意图

毛刺信号通过定义其宽度、高度和面积 3 个变量来进行量化，如图 7-21 所示。

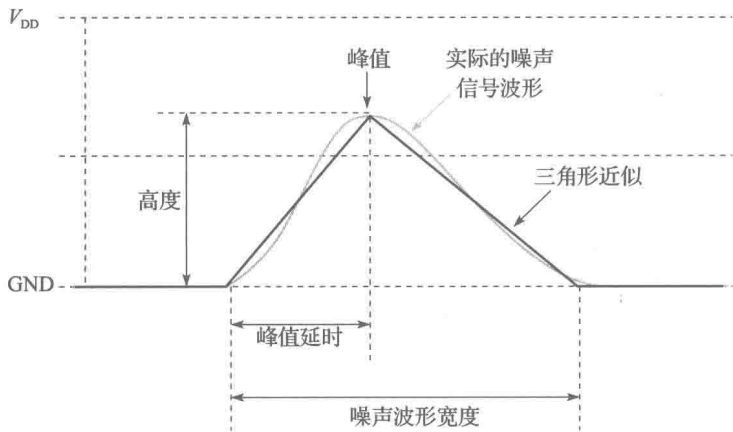


图 7-21 毛刺信号量化

高度的大小为毛刺信号最高电压峰值，宽度大小为毛刺信号脉冲宽度值，面积大小为毛刺信号波形所覆盖的电压区域面积。

对应不同的输出负载，噪声电平以不同的宽度与高度表示，如图 7-22 所示。

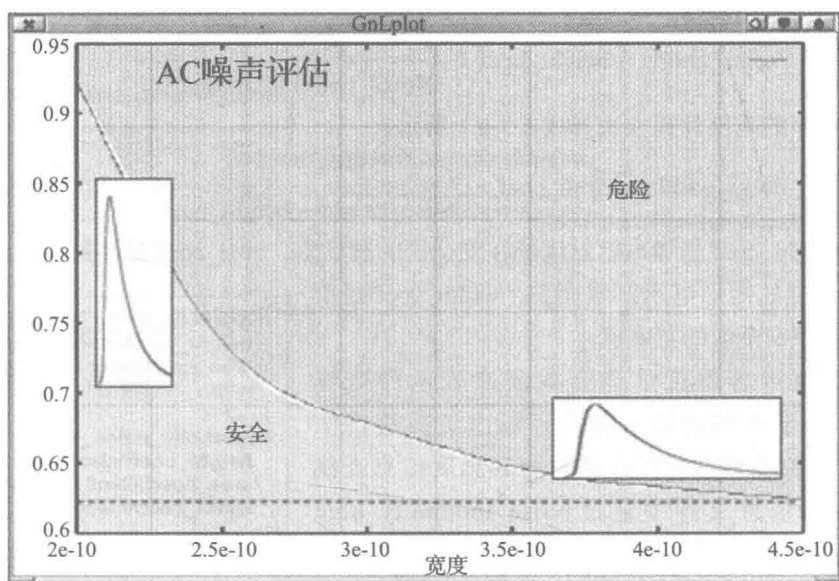


图 7-22 噪声电平信息图

如图 7-22 所示，噪声容忍度分为安全 (safe) 和危险 (hazard) 两个区域，当毛刺电平值落在在安全区域时，逻辑门单元工作正常，如果毛刺电平值落在危险区域时，则会对逻辑门单元的性能造成不利影响甚至出现逻辑功能错误，如图 7-23 所示。

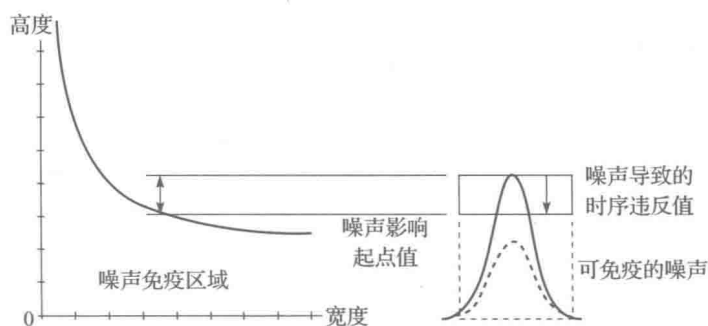


图 7-23 噪声容忍度

噪声容忍度的仿真结果一般可以通过两种形式保存在时序信息文件中：

1) 通过双曲线噪声曲线来表示噪声容忍度信息，在时序信息文件中通过 `hyperbolic_noise_high` 和 `hyperbolic_noise_low` 这 2 个双曲线噪声组来定义。基于双曲线噪声组中保存的

信息可以计算出毛刺电平曲线的高度和宽度信息,如图 7-24 所示。

双曲线噪声曲线作用在逻辑门单元的输入端口,并且基于独立的低负载值进行仿真模拟得到双曲线噪声组的信息。

双曲线噪声的高度计算方法如式(7-5)所示。

$$\text{Noise_height} = \text{height_coef} + \frac{\text{area_coef}}{\text{Noise_width} - \text{width_coef}} \quad (7-5)$$

双曲线噪声的宽度计算方法如式(7-6)所示。

$$\text{noise_width} = \text{width_coef} + \frac{\text{area_coef}}{\text{noise_height} - \text{height_coef}} \quad (7-6)$$

其中, height_coef 指噪声容忍度处在安全区的高度值, area_coef 是毛刺信号波形超出或低于噪声容忍度的矩形电压区域面积, width_coef 是指噪声容忍度处在安全区的宽度值。

2) 通过噪声容忍度信息查找表来表示噪声容忍度信息,基于不同的输入噪声和输出负载进行仿真,并把对应的毛刺信息保存在噪声容忍度信息查找表中。在时序信息文件中通过 noise_immunity_high 和 noise_immunity_low 这 2 个噪声容限组来定义,如图 7-25 所示。

```
hyperbolic_noise_high() {
  height_coefficient : 0.380186932375;
  area_coefficient : 0.843980211331;
  width_coefficient : 0;
}
hyperbolic_noise_low() {
  height_coefficient : 0.0427488335888;
  area_coefficient : 0.743492709403;
  width_coefficient : 0;
}
```

图 7-24 双曲线噪声曲线信息

```
noise_immunity_high(ni_inw_oload_7x5) {
  index_1("0, 0.01, 0.2865868, 1.257467, 3.111503, 6.0, 6.6"); <- noise width
  index_2("0.0, 0.015574, 0.07024583, 0.174648, 0.3373014"); <- output load
  values("6.48, 6.48, 6.48, 6.48, 6.48", \<- noise height
    "1.62, 1.62, 1.62, 1.62, 1.62", \<- noise height
    ....
  )
}

noise_immunity_low(ni_inw_oload_7x5) {
  index_1("0, 0.01, 0.2865868, 1.257467, 3.111503, 6.0, 6.6");
  index_2("0.0, 0.00776, 0.03503891, 0.0871151, 0.1682473");
  values("6.48, 6.48, 6.48, 6.48, 6.48", \
    "1.62, 1.62, 1.62, 1.62, 1.62", \
    ....
  )
}
```

图 7-25 噪声容限组信息

噪声传播以噪声增强与衰减的形式表示,对于不同负载、不同输入信号,噪声的影响也不同,如图 7-26 所示。

在时序信息文件中通过 propagated_noise_height_high、propagated_noise_height_low、propagated_noise_width_high 和 propagated_noise_width_low 这 4 个噪声传播组来定义,如图 7-27 所示。

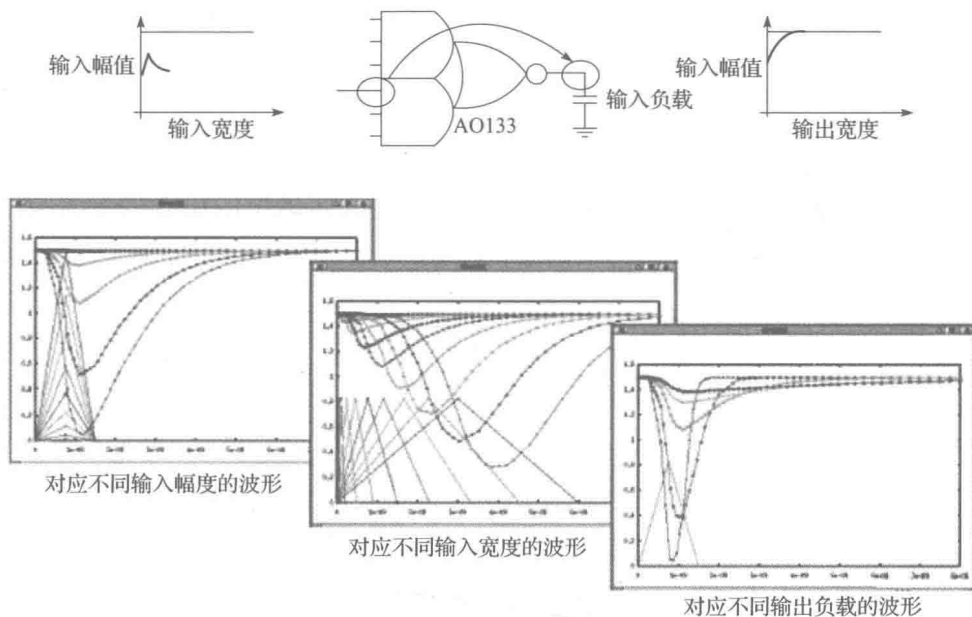


图 7-26 噪声传播信息图

```

propagated_noise_height_high(pn_inw_inh_oload_6x9x5) {
  Index_1 ("0, 0.01, 0.2865868, 1.267467, 3.111503, 6.0"); <- input noise width
  Index_2 ("0, 0.324, 0.4628571428, 0.6017142857, 0.7405714285, 0.8794228571,
1.0182815714, 1.1571428571, 1.296, 1.62"); <- input noise height
  Index_3 ("0.0, 0.01557481, 0.07024583, 0.174648, 0.3373014"); <- output load
  values {"...", \
    "...", \
    "...", \
  }
  propagated_noise_height_low(pn_inw_inh_oload_6x9x5) {
    ....
  }
}

```

图 7-27 噪声传播组信息

7.7 噪声延时计算方法

在静态时序分析中，噪声影响会以延时的形式累加在正常的时序路径延时上。前面已经介绍过，基于不同的情况，噪声可以对信号传播延时起到加速和减速的作用。处于噪声容忍

度的安全区域的噪声可以加速信号传播延时，相反，处于噪声容忍度的危险区域的噪声会减小信号传播延时，而处在安全区域和危险区域边界上的噪声不对信号传播产生影响。噪声容忍度区域图如图 7-28 所示。

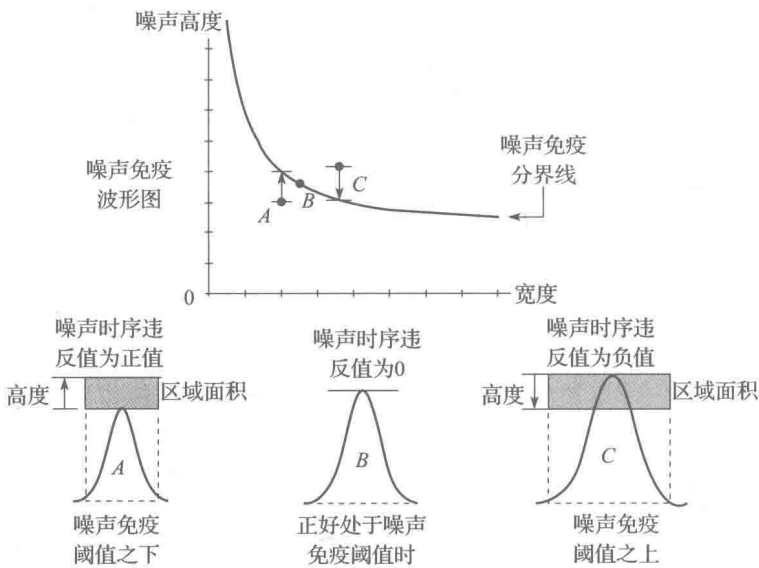


图 7-28 噪声容忍度区域图

目前，时序分析工具通过以下 3 种方法来计算噪声带来的延时。

(1) 基于面积的方法

基于面积的方法中的面积是指毛刺信号波形超出或低于噪声容忍度的矩形电压区域面积。该面积的值等于毛刺信号波形的峰值电压与边界电压的差，乘以毛刺信号波形的脉宽。其中边界电压即为安全区域和危险区域边界上的电压值，面积的单位为电压 × 时间，比如 mV × ns。基于面积的噪声延时计算方法如图 7-29 所示。

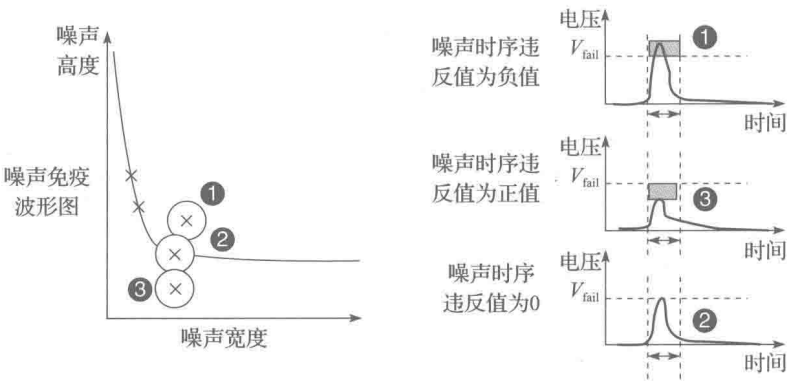


图 7-29 基于面积的噪声延时计算

如图 7-29 所示, 位于节点 1 的毛刺信号产生了负的延时值, 并累加到了信号传播延时中, 从而减小了信号传播延时。位于节点 3 的毛刺信号产生了正的延时值, 并累加到了信号传播延时中, 从而加速了信号传播延时。位于节点 2 的毛刺信号处于边界电压上, 因此对信号传播延时不产生影响。

(2) 基于高度的方法

基于高度的方法中的高度是指毛刺信号波形峰值电压超出或低于噪声容忍度的电压值, 即电压差。基于高度的噪声延时计算方法如图 7-30 所示。

基于高度的方法与基于面积的方法的区别是, 在计算噪声的延时, 前者只关注毛刺信号波形峰值电压与边界电压的电压差, 不考虑毛刺信号的脉宽因素。

(3) 基于面积百分比的方法

基于面积百分比的方法是通过计算出毛刺信号波形超出或低于噪声容忍度的矩形电压区域面积与对应安全区域的面积之比后, 再计算出噪声延时。基于面积的噪声延时计算方法如图 7-31 所示。

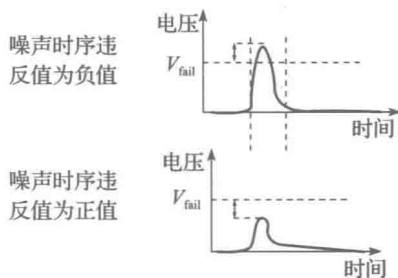


图 7-30 基于高度的噪声延时计算

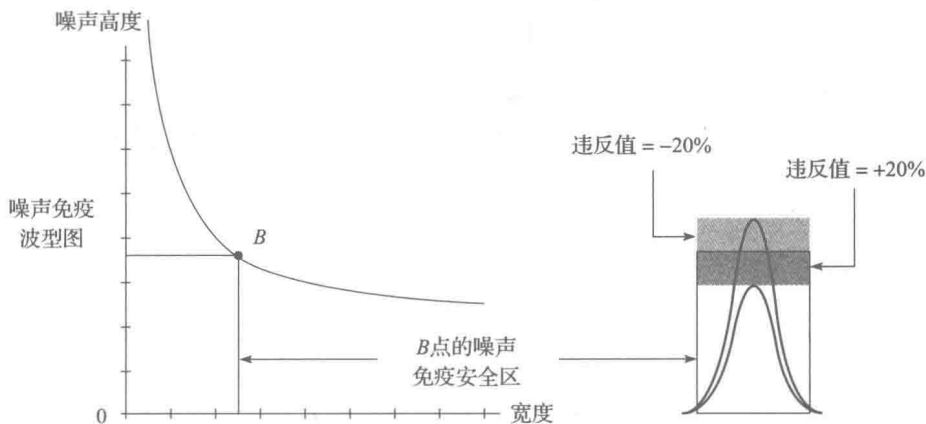


图 7-31 基于面积的噪声延时计算

如图 7-31 所示, 毛刺信号波形的延时乘以面积百分比值即为要累加到信号传播延时上的噪声延时, 其值的正负根据毛刺信号波形来确定, 如果位于安全区域则为正, 相反为负。

7.8 时间窗口

串扰噪声只有在受扰线和干扰线信号同时转换的时候才会影响受扰线的传播延时, 因此时序分析工具可以引入时间窗口的分析方法对串扰带来的延时进行精确计算。

单元输出端的信号变化取决于该单元所有输入端的时序路径输出信号，因此单元输出转换的最快时间点和最慢时间点取决于输入端时序路径的最快和最慢路径，而输出端进行转换最快时间点到最慢时间点这个时间段就是时间窗口（timing window）。时间窗口如图 7-32 所示。

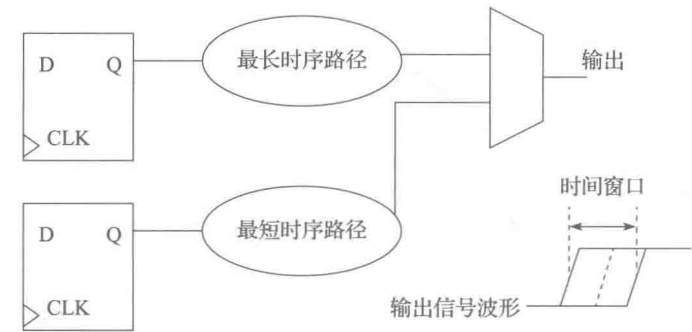


图 7-32 时间窗口

那么重叠时间窗口就是干扰线时间窗口与受扰线时间窗口的重叠区域，如图 7-33 所示。

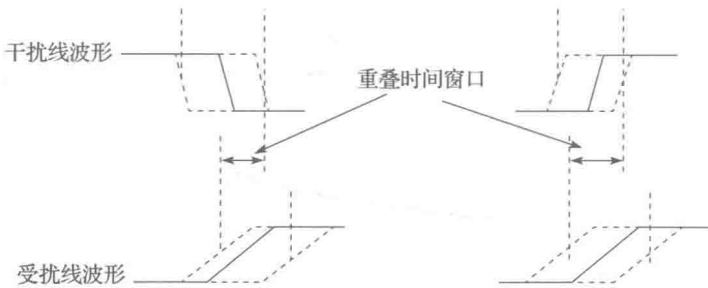


图 7-33 重叠时间窗口

下面是通过时间窗口方法计算串扰噪声带来额外延时的简单例子，如图 7-34 所示。

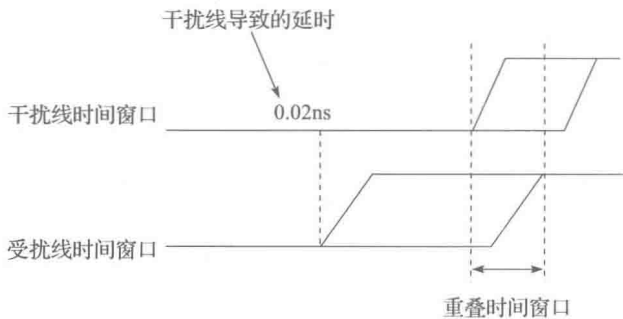


图 7-34 时间窗口计算方法

由于干扰线与受扰线存在重叠时间窗口，同时由于串扰噪声，干扰线给受扰线带来的额

外延时为 0.02ns ，那么该受扰线的输出延时就增加了 0.02ns 。

对于多条干扰线作用于同一条受扰线的情况，可能会存在时间窗口重叠，但其分析方法类似，并且串扰延时结果选取最坏的串扰延时用于时序分析，如图 7-35 所示。

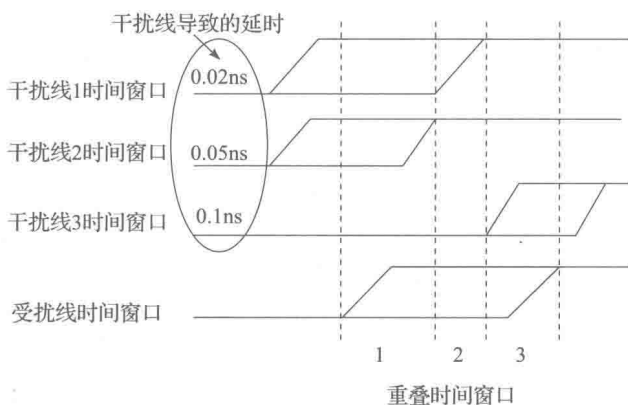


图 7-35 时间窗口重叠计算方法

由于 3 条干扰线的时间窗口与受扰线时间窗口存在 3 个重叠的时间窗口，分别为干扰线 1 和干扰线 2 与受扰线的重叠时间窗口 1、干扰线 1 与受扰线的重叠时间窗口 2 和干扰线 3 与受扰线的重叠时间窗口 3，所以对 3 个重叠窗口带来的串扰噪声额外延时分别进行计算，如下所示。

$$\begin{aligned}\text{串扰噪声延时（重叠时间窗口 1）} &= \text{干扰线 1 串扰延时} + \text{干扰线 2 串扰延时} = 0.02\text{ns} + 0.05\text{ns} \\ &= 0.07\text{ns}\end{aligned}$$

$$\text{串扰噪声延时（重叠时间窗口 2）} = \text{干扰线 1 串扰延时} = 0.02\text{ns}$$

$$\text{串扰噪声延时（重叠时间窗口 3）} = \text{干扰线 3 串扰延时} = 0.1\text{ns}$$

前面讲到，串扰延时结果选取最坏的串扰延时用于时序分析，因此最坏的串扰延时为重叠时间窗口 3 导致的串扰延时，其值为 0.1ns 。

根据干扰线与受扰线的信号变化方向，其串扰噪声影响延时可以分为延迟和加速两种结果，所以串扰延时并不唯一只增加受扰线的额外延时，是加速还是延迟，要根据设计的实际情况而定。

7.9 优化噪声的物理方法

随着流片工艺尺寸越来越小，串扰噪声问题很难被根本解决。这里所要探讨的是尽量将串扰噪声问题限制在可控范围内，并努力减少后期修复的迭代次数。防止和修复的目的则在于当发现串扰问题时采取适当手段减小串扰噪声危害。

以下列出几种常用的避免及修复串扰的方法。

1) 增大受扰信号线与干扰信号线的间距, 如图 7-36 所示。

由于导致噪声的耦合电容与线间距的平方成反比, 因此增大连线间距是减少噪声最直接、最有效的方式。但是由于芯片面积的限制, 这种方法无法大面积使用, 默认情况下只在部分线网 (如时钟信号) 上才会采用。

2) 在受扰信号线上加入屏蔽线, 如图 7-37 所示。



图 7-36 增大间距

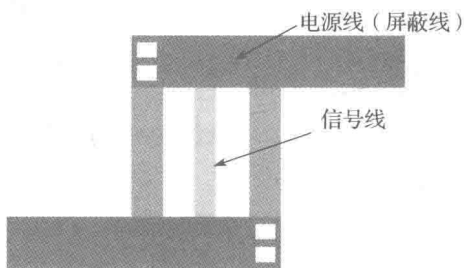


图 7-37 加入屏蔽线

在同一层中, 除了可以增大线间距, 还可以在连线间插入屏蔽线 (GND 或 VDD), 以减小耦合电容噪声。而在信号线层间插入 GND 或 VDD 金属层也能有效地把线间的耦合电容转换为对地电容, 从而消除干扰。

3) 在受扰信号线上插入缓冲器, 如图 7-38 所示。

干扰线向受扰线注入耦合脉冲, 因为耦合电容正比于并行连线长度, 所以如果并行长度超过一定程度以后, 产生的串扰噪声很容易超过噪声容限, 从而引起功能紊乱。在受扰线上插入一个缓冲器, 将耦合电容分摊在两端连线上, 这样在缓冲器输入端就获得较小的噪声。虽然插入缓冲器技术的确可以减小功能噪声出现的几率, 但因为多了一级门, 可能会对路径的延迟有一定影响。

4) 调整受扰信号的或者干扰信号的驱动单元强度, 如图 7-39 所示。

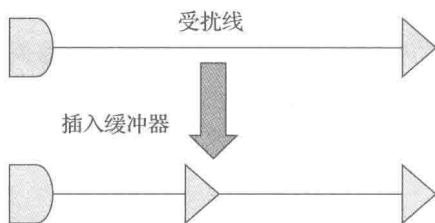


图 7-38 插入缓冲器

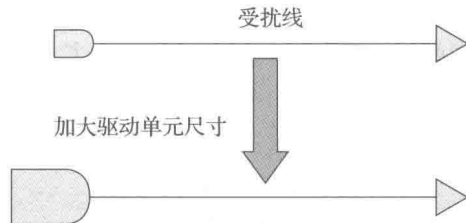


图 7-39 加大驱动单元尺寸

增强受扰线驱动端的驱动强度和减小干扰线驱动端的驱动强度都可以缓解串扰噪声的影响。但问题在于如何区分干扰线和受扰线, 因为在许多设计中, 一条连线可能既是干扰线同时又是受扰线, 通过增强驱动强度来减小产生在其上的串扰噪声, 可能会导致增加其注入其他连线的噪声。

5) 通过改变走线形式减少并行线距离, 如图 7-40 所示。

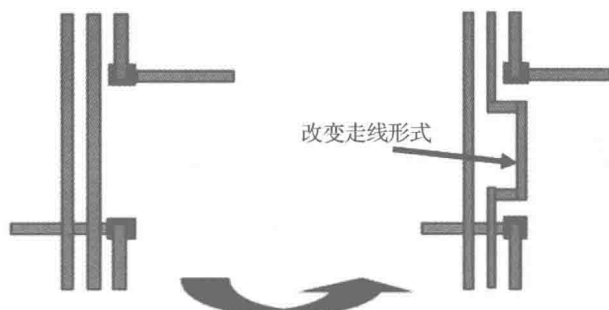


图 7-40 改变走线形式

干扰线向受扰线注入耦合脉冲, 因为耦合电容正比于并行连线长度, 所以如果并行长度超过一定程度, 产生的串扰噪声很容易超过噪声容限, 引起功能紊乱。可以通过改变同层金属走线形式的方法来减少由于过长并行连线导致的大耦合电容的形成。

也可以把同层并行太长的互连线跳层改变成不同层的走线形式来减少耦合电容, 如图 7-41 所示。

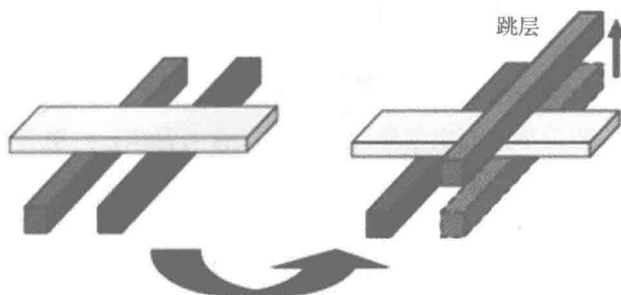


图 7-41 互连线跳层

6) 尽量减少受扰线的走线长度, 如图 7-42 所示。

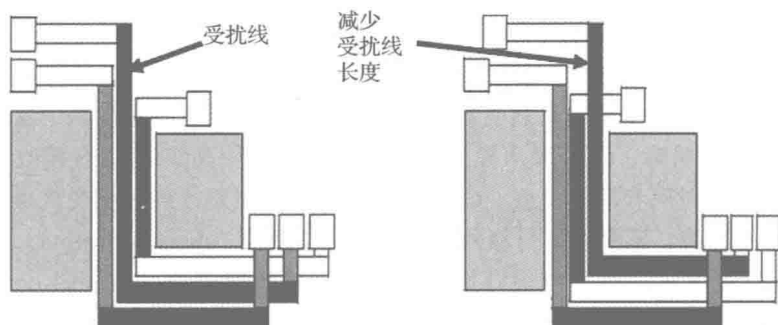


图 7-42 减少受扰线长度

除了减少并行线长度可以减少耦合电容减低噪声影响以外，直接减少受扰线的走线长度也能够减少其被噪声影响的概率。优先完成受扰线的逻辑连接，这样可以用最短的走线路径完成连接。

7.10 CCS 噪声模型

CCS 噪声模型是一种高级的、基于电流的驱动模型，能够实现准确的噪声分析，其结果非常接近 SPICE 模拟仿真结果。它不仅能够对串扰噪声进行精确的建模，还能对传播的噪声冲击（propagated noise bump）和驱动衰减（driver weakening）进行分析。基于 CCS 噪声模型，时序分析工具能够在 65nm 及以下的先进工艺下对噪声进行精确的分析。

由于 CCS 噪声模型是一种高级的电流驱动模型，因此能获得单元的静态和瞬态特性。CCS 噪声模型的静态特性信息由一个电流查找表构成，该查找表储存的是输入和输出电压水平的函数结果，它可以通过一种类似基本的 V_{in}/V_{out} ($ViVo$) 模型的高效率直流分析获得。

CCS 噪声模型的关键优势是它采用几种动态参数来对单元的动态响应建模，这是静态电流表所不能获得的。动态参数从瞬态分析测量中获取，瞬态分析测量记录了单元对某个输入变化和噪声冲击的响应。

CCS 噪声模型可以准确地对所有的串扰噪声分析效应进行建模，包括噪声计算、噪声传播、驱动衰减以及组合噪声传播和噪声输入。在噪声分析中，噪声计算是指干扰网络的输入噪声冲击特性的计算，并假设受扰信号驱动本身不产生噪声。噪声传播是指噪声冲击信号通过电路单元的传播，并假设单元输出信号网络之间没有电容耦合。组合噪声传播和输入噪声分析是指一般情况下在受扰驱动信号中的传播噪声，以及与受扰网络对应的干扰网络的噪声输入分析。驱动衰减可以认为是噪声组合的一种特殊情况，这种情况下，传播的噪声本身很小，但是因为增大了输入信号噪声冲击，它大幅度降低了受扰驱动的有效驱动强度。广泛研究表明，对于以上这些串扰噪声分析来说，CCS 噪声模型比其他模型的准确性高得多，主要是因为 CCS 噪声模型的动态参数。CCS 噪声模型与其他噪声模型计算结果的对比如图 7-43 所示。

如图 7-43 所示为使用 CCS 噪声模型计算出的传播噪声波形。其中，下方曲线是输入噪声波形；上方实线曲线是 SPICE 输出波形；上方虚线是使用噪声模型运算出来的输出波形。它与 SPICE 波形的匹配程度比没有使用动态参数的基本 $ViVo$ 驱动模型（图 7-43）要精确得多。

准确的 CCS 噪声分析流程中还需要一种接收器模型，这种模型能获得在输入转换时的接收器引脚有效电容负载与输出负载电容负载的相关度。这样比时序模型中常见的只有单一值的引脚电容模型要精确。CCS 噪声模型分析流程可以利用高级的 CCS 接收器模型，这种模型依赖于输入转换时间和输出负载。CCS 噪声模型分析引擎已经包括受扰接收器中变化的有效输入电容器的影响，不需要进行额外的接收器特性提取（characterization）。

基于 $ViVo$ 的电流驱动模型能很好地获得单个连接通道模块（channel-connected block, CCB）的行为。对于具有超过一个 CCB 的复杂电路单元，电路单元的晶体管级网表需要划分成多个 CCB，每个 CCB 对应一个 CCS 噪声模型。晶体管级网表划分和 CCS 噪声模型参数

获取在单元时序特性提取的阶段来实现。

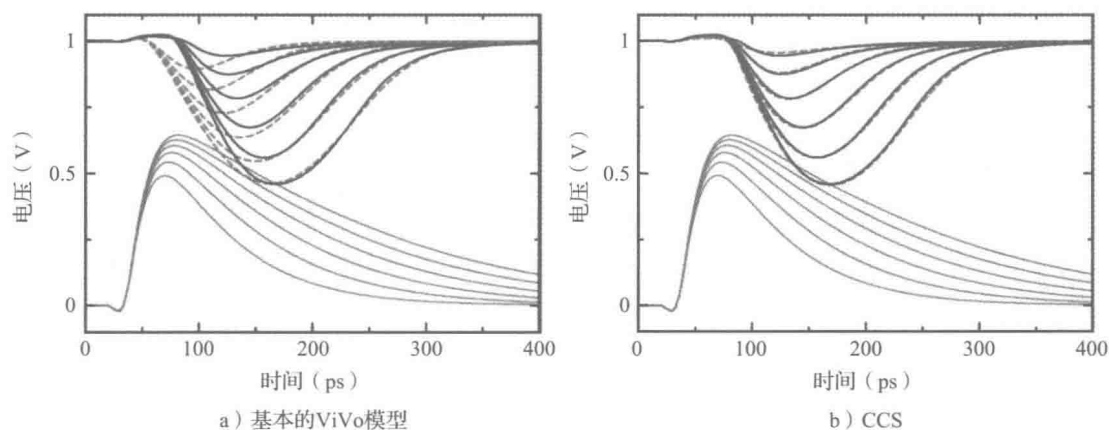


图 7-43 噪声模型结果比较

CCS 噪声模型是一个基于电流的单元边界晶体管级 (boundary transistor stage) 模型, 无需对单元内的每一个晶体管级进行建模, 如图 7-44 所示。

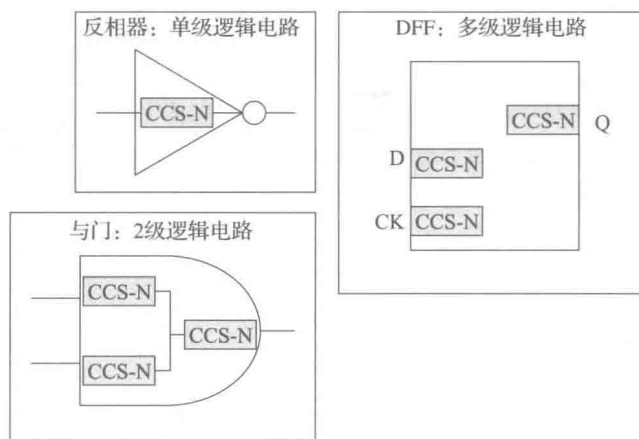


图 7-44 CCS 噪声边界晶体管级建模技术

如图 7-44 所示, 对于单级逻辑电路 (如反相器), 整个 CCB 级均在 CCS 噪声格式时序弧内得到了描述。如果是 2 级的门电路 (如与门), 要对 2 个相连的 CCB 级进行时序建模。对于超过 2 级 CCB 的单元, 采用对单元输入引脚和输出引脚进行时序建模的方法进行噪声分析。因此, 单元的时序特性提取一旦完成, CCS 噪声模型数据就被保存在时序弧上或者单元库中的一个引脚信息上, 这取决于电路网表的拓扑结构。

因此, 对于输入输出引脚对只有一个 CCB 的电路单元, 只需要得到一个 CCS 噪声模型, 并保存在时序弧中。这样的单级 CCB 单元包含了大部分的反相器单元、与非门单元、

或门单元、AOI 单元、OAI 单元等。对于那些具有两个连续的 CCB 的电路单元，两个 CCS 噪声模型被保存在时序弧中。这样的二级 CCB 单元包括大部分的缓冲器单元、与门单元、或门单元、与或门单元和或或门单元等。对于具有 3 个或更多的 CCB 的电路单元，包含大部分的触发器单元、全加器单元、数字宏模块等，CCS 噪声模型数据被储存在引脚信息上。

CCS 噪声模型相关定义

当对基于 CCS 噪声模型进行噪声分析时，时序信息文件中需要使用与 CCS 噪声模型相关的组和属性。下面只部分介绍 CCS 噪声模型中比较重要的组和属性。关于 CCS 噪声模型文件的详细内容可以参考附录中 CCS 时序信息文件片段内容。

1. CCS 噪声前级组

CCS 噪声前级组 (ccsn_first_stage group) 是 CCS 噪声模型中用于定义第一级连接通道模块的噪声信息。CCS 噪声前级组可以定义在单元组或者引脚组中，不同的逻辑门单元对应自己的 CCS 噪声前级组。定义一个名为 CLK 引脚的 CCS 噪声前级组如下所示：

```
pin (CLK) {
    .....
    ccsn_first_stage () {
        .....
    }
}
```

CCS 噪声前级组的内容里存在需求属性 (is_needed attribute)、反向属性 (is_inverting attribute)、级数属性 (stage_type attribute)、上拉米勒电容属性 (miller_cap_rise attribute) 和下拉米勒电容属性 (miller_cap_fall attribute) 等，这里对主要的属性进行说明。

(1) 需求属性

需求属性是 CCS 噪声前级组中用于定义是否需要储存连接通道模块噪声信息的属性。该属性定义的示例如下。

```
is_needed : true;
```

(2) 反向属性

反向属性是 CCS 噪声前级组中用于定义连接通道模块的噪声是否反向的属性。该属性定义的示例如下。

```
is_inverting : true;
```

(3) 级数属性

级数属性是 CCS 噪声前级组中用于定义连接通道模块的噪声输出电压是上拉、下拉还是都存在的属性。该属性定义的示例如下。

```
stage_type : "both" ;
```

(4) 上拉米勒电容属性

上拉米勒电容属性是 CCS 噪声前级组中用于定义连接通道模块的噪声输出电平为上拉时的米勒电容值属性。该属性定义的示例如下。

```
miller_cap_rise : 0.273328;
```

(5) 下拉米勒电容属性

下拉米勒电容属性是 CCS 噪声前级组中用于定义连接通道模块的噪声输出电平为下拉时的米勒电容值属性。该属性定义的示例如下。

```
miller_cap_fall : 0.256533;
```

(6) 输出信号电平属性

输出信号电平属性 (output_signal_level attribute) 是 CCS 噪声前级组中用于定义连接通道模块的噪声输出电平的电压名称属性。该属性定义的示例如下。

```
output_signal_level : VDD1;
```

(7) 输入信号电平属性

输出信号电平属性 (input_signal_level attribute) 是 CCS 噪声前级组中用于定义连接通道模块的噪声输入电平的电压名称属性。该属性定义的示例如下。

```
input_signal_level : VDD2;
```

2. CCS 噪声末级组

CCS 噪声末级组 (ccsn_last_stage group) 是 CCS 噪声模型中用于定义第二级连接通道模块的噪声信息。CCS 噪声末级组可以定义在单元组或者引脚组中, 不同的逻辑门单元对应自己的 CCS 噪声末级组。定义一个名为 CLK 引脚的 CCS 噪声后级组如下所示:

```
pin (CLK) {
    .....
    ccsn_last_stage () {
        .....
    }
}
```

CCS 噪声末级组的内容里存在的属性和 CCS 噪声前级组一样, 这里不重复说明。

3. 直流电流组

直流电流组 (dc_current group) 是 CCS 噪声模型中用于定义根据对应的输入和输出电压值所对应的连接通道模块的噪声电流信息查找表。不同的逻辑门单元对应自己的直流电流组, 如下所示:

```
dc_current (dc_current_name) {
    index_1( "float, ..." );
    index_2( "float, ..." );
    values( "float, ..." );
}
```

如上所示，与延时查找表模板组结构类似，直流电流组也通过两个索引变量 `index_1` 和 `index_2` 的值来确定最终的噪声电流结果。

4. 输出高电平组

输出高电平组 (`output_voltage_rise group`) 是 CCS 噪声模型中用于定义连接通道模块的噪声输出高电平时电压信息的组。不同的逻辑门单元对应自己的输出高电平组，如下所示：

```
ccsn_last_stage () {
.....
output_voltage_rise () {
.....
}
}
```

5. 输出低电平组

输出高电平组 (`output_voltage_fall group`) 是 CCS 噪声模型中用于定义连接通道模块的噪声输出低电平时电压信息的组。不同的逻辑门单元对应自己的输出低电平组，如下所示：

```
ccsn_last_stage () {
.....
output_voltage_fall () {
.....
}
}
```

6. 高电平噪声传播组

高电平噪声传播组 (`propagated_noise_high group`) 是 CCS 噪声模型中用于定义连接通道模块的高电平传播噪声电压信息的组。不同的逻辑门单元对应自己的高电平噪声传播组，如下所示：

```
ccsn_last_stage () {
.....
propagated_noise_high () {
.....
}
}
```

7. 低电平噪声传播组

低电平噪声传播组 (`propagated_noise_low group`) 是 CCS 噪声模型中用于定义连接通道模块的低电平传播噪声电压信息的组。不同的逻辑门单元对应自己的低电平噪声传播组，如下所示：

```
ccsn_last_stage () {
.....
propagated_noise_low () {
.....
}
}
```

第 8 章

单元时序建模实战

8.1 时序信息提取实现

如果自行设计了时序单元 HLFF 的原理图和版图，但是在设计中时序分析工具需要读取该单元的时序信息，如综合工具需要知道单元的逻辑功能、单元实际的输入负载电容、在不同输入斜率和输出负载情况下单元的延时和功耗、单元的面积等。单元时序信息特征化就是用模拟仿真器来提取标准单元以上信息的过程。通过时序信息特征化来提供单元的时序数据给多种时序分析工具使用，本章中对单元的时序信息特征化所使用的 EDA 工具为 Cadence 公司的 SignalStorm Library Characterizer，业界简称 SLC，提取的单元实例为时序单元 HLFF。

8.1.1 时序信息特征化实现流程

SignalStorm Library Characterizer 可提取标准单元（数字的）和 I/O 端口的时序信息，并将其整合为 .lib 文件。典型特征化流程如图 8-1 所示。

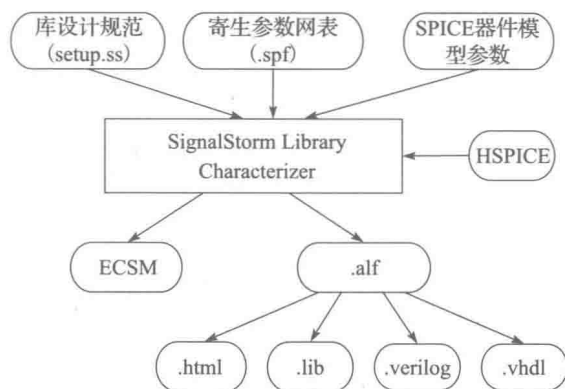


图 8-1 标准单元特征化流程

如图 8-1 所示，特征化流程中将库设计规范文件、标准单元的带寄生参数网表文件及 Hspice 器件仿真模型参数文件输入 SignalStormLC 中，内部机制调用电路仿真器软件（如

Hspice) 进行模拟, 可得到标准单元的有效电流源模型文件 (ECSM) 及中间文件 (.alf), 中间文件 (.spf) 可转化为多种数据格式 (如 .html、.lib、.v 或 .vhdl), 供其他时序分析工具使用。

8.1.2 时序信息特征化数据准备

在使用 SignalStorm Library Characterizer 工具进行时序信息特征化工作之前, 除了准备标准单元的带寄生参数网表文件及 Hspice 器件仿真模型参数文件 (这些数据都是来自前期电流设计阶段已经存在的设计数据) 以外, 还需要准备正确的库设计规范文件和单元属性定义文件 (property.def), 这两个文件是 SLC 工具自身特有的输入文件。本节主要介绍如何准备用于时序单元 HLFF 时序特征化实现所需要的库设计规范文件和单元属性定义文件。

1. 库设计规范文件

setup.ss 文件是 SLC 提取时序信息 (.lib) 必备的输入文件, 这里以在典型情况下, 时序单元 HLFF 时序特征化实现使用的 setup.ss 为例进行介绍。setup.ss 文件主要包括设计参数定义设置和过程控制设置两部分, 设计参数定义设置部分定义了必备的工艺信息、单元的输入斜率和输出负载等信息, 过程控制设置部分定义了不同工艺下时序特征化时所调用的设计参数定义设置部分中的设计参数。

设计参数定义设置主要包括工艺参数定义、信号参数定义、模拟参数定义、索引参数定义、组参数定义等。过程控制设置主要包括工艺选择设置、索引选择设置和组选择设置等。下面基于典型工艺下对以上部分重要信息定义的格式及内容进行进一步的介绍。

(1) 工艺参数定义

工艺参数定义包含时序特征化时对应工艺下的电压、温度和阈值等信息, 不同工艺对应不同的工艺参数定义内容。

工艺参数定义的格式如下所示。

```
Process typical{
voltage=1.2;
temp=25;
Corner="tt";
Vtn=0.402;
Vtp=0.395;
};
```

以上内容为用于时序单元 HLFF 时序特征化实现的工艺参数定义, 通过该工艺参数定义设置了在典型工艺下的电压值为 1.2V, 温度为 25℃, NMOS 晶体管的阈值电压为 0.402V, PMOS 晶体管的阈值电压为 0.395V。工艺参数的具体数值来源于仿真模拟参数文件, 该模型文件由流片厂家提供。

(2) 信号参数定义

信号参数定义包含输入信号高电平和低电平及高、低跳变斜率电压等信息。

信号参数定义的格式如下所示。

```
Signal std_cell {
unit=REL;
Vh=1.0 1.0;
Vl=0.0 0.0;
Vth=0.5 0.5;
Vsh=0.7 0.7;
Vsl=0.3 0.3;
tsmax=3.0n;
};
```

以上内容为用于时序单元 HLFF 时序特征化实现的信号参数定义,该信号参数定义设置了度量单位为百分比,高电平为电源电压值的 100%,低电平为 0,信号阈值电压为电源电压的 50%,信号逻辑 1 的最高电压为电源电压的 70%,信号逻辑 0 的最低电压为电源电压的 30%,输出信号最大转换时间为 3ns。

(3) 模拟参数定义

模拟参数定义包含时序特征化时直流分析条件和二分法搜索条件。

模拟参数定义的格式如下所示。

```
Simulation std_cell{
transient=0.02n 60n 2p;
dc=0.26 1.5 0.01;
bisec=12.0n 6.0n 2p;
resistance=10MEG;
};
```

以上内容为用于时序单元 HLFF 时序特征化实现的模拟参数定义,该模拟参数定义设置了瞬态分析在 20ps ~ 60ns 的时段里,以 2ps 的间隔进行扫描仿真,直流分析从 0.26V 到 1.5V 的范围内,以 0.01V 的间隔进行扫描仿真,二分法搜索分析在 6 ~ 12ns 区间内以 2ps 为最小单位,对保持时间和建立时间进行搜索仿真,用于 3 态模拟的上拉和下拉电阻值为 10M Ω 。

(4) 索引参数定义

索引参数定义包含单元库中 1 倍、2 倍、3 倍、4 倍、8 倍等的输入跳变及输出负载值,不同工艺时序库对应不同索引参数定义的跳变斜率和负载值。

索引参数定义的格式如下所示。

```
Index X1 {
Bslew=0.0375n 1.875n 3.75n;
Slew=0.0375n 0.125n 0.5n 1.125n 1.875n 2.75n 3.75n;
Load=0.00035p 0.021p 0.0385p 0.084p 0.147p 0.231p 0.3115p;
};
Index XL{ Load=0.000175p 0.0105p 0.01925p 0.042p 0.0735p 0.1155p 0.15575p;};
Index X2{ Load=0.0007p 0.042p 0.077p 0.168p 0.294p 0.462p 0.623p;};
Index X3{ Load=0.00105p 0.063p 0.1155p 0.252p 0.441p 0.693p 0.9345p;};
Index X4{ Load=0.0014p 0.084p 0.154p 0.336p 0.588p 0.924p 1.246p;};
Index X8{ Load=0.0028p 0.168p 0.308p 0.67p 1.176p 1.848p 2.492p;};
Index CLK_Slew {Bslew=0.0475n 1.975n 2.95n;};
```

```
Index EN_Slew {Bslew=0.0475n 1.975n 2.95n};
```

以上内容为用于时序单元 HLFF 时序特征化实现的索引参数定义, 该索引参数定义设置了二分法搜索分析中的信号转换时间, 用于时序文件中 7×7 查找表的信号转换时间, 用于时序文件中 7×7 查找表的输出负载值。其中第一个定义的索引参数为默认设置, 后面定义了改变的索引参数, 如后续的索引参数主要是在负载值和信号转换时间上有变化, 因此只需要设置负载值和信号转载时间的索引参数。

(5) 组参数定义

组参数定义为归纳具体的设计参数适用于哪些标准单元或者输入输出引脚。

组参数定义的格式如下所示。

```
Group X1 {CELL=HLFF};
Group CLK_Slew {PIN=*.CLK};
```

以上内容为用于时序单元 HLFF 时序特征化实现的组参数定义, 该组参数定义设置了时序单元 HLFF 使用索引 $\times 1$ 的参数进行模拟仿真, 同时其时钟输入端 CLK 调用索引 CLK_Slew 的参数进行二分法搜索仿真。

(6) 裕度参数定义

裕度参数定义设置了时序特征化结果比实际仿真结果之间的余量值。

裕度参数的格式如下所示。

```
Margin m0{
    cap = 1.05 0.0;
    delay = 1.05 0.0;
    power = 1.05 0.0;
    setup = 1.05 0.0;
    hold = 1.05 0.0;
    release = 1.05 0.0;
    removal = 1.05 0.0;
    recovery = 1.05 0.0;
    width = 1.05 0.0;
};
```

以上内容为用于时序单元 HLFF 时序特征化实现的裕度参数定义, 该裕度参数定义设置了 HLFF 在时序特征化的结果比实际仿真的结果在延时、功耗、负载等参数存在 5% 的余量值。

(7) 平均参数定义

平均参数定义设置了通过模拟仿真结果的最大值和最小值来求出平均值的比例因子。

平均参数定义格式如下所示。

```
Nominal n0 {
    cap = 0.5 0.5;
```

```

delay = 0.5 0.5;
power = 0.5 0.5;
};

```

以上为用于时序单元 HLFF 时序特征化实现的平均参数定义，该平均参数定义设置比例因子为 0.5。

(8) 工艺选择设置

工艺选择设置定义了对应工艺下使用对应的设计参数。

工艺选择设置格式如下所示。

```

set process(typical){
simulation = std_cell;
signal = std_cell;
margin = m0;
nominal = n0;
index = X1;
load =load_X1;
};

```

以上内容为用于时序单元 HLFF 时序特征化实现的工艺选择设置，该工艺选择设置定义 typical 工艺下使用设计参数 std_cell、m0、n0 和 X1 设置，同时该设置定义为默认设置。

(9) 索引选择设置

索引选择设置定义了对应工艺下使用对应的组参数。

索引选择设置格式如下所示。

```

set index(typical){
Group(XL) = XL;
Group(X1) = X1;
Group(X2) = X2;
Group(X3) = X3;
Group(X4) = X4;
Group(X8) = X8;
Group(CLK_Slew) = CLK_Slew;
Group(EN_Slew) = EN_Slew;
};

```

以上内容为用于时序单元 HLFF 时序特征化实现的索引选择设置，该索引选择设置定义了 typical 工艺下使用组参数 XL、X1 和 CLK_Slew 等设置。

2. 单元属性定义文件

单元属性定义文件内容包含了时序特征化单元的简称（footprint）定义、IO PAD 属性定义和端口定义，端口定义中除了常规的输入输出端口以外，还包括具有扫描功能单元的测试扫描端口。

用于时序单元 HLFF 时序特征化实现的单元属性定义文件如下所示：

```
cell HLFF {
  pin D{
    direction input;
  };
  pin CLK{
    direction input;
  };
  pin Q{
    direction output;
  };
  footprint hlff;
};
```

以上内容定义了时序单元 HLFF 的输入输出端口及简称定义。

完成库设计规范文件和单元属性定义文件准备好后，再加上 Hspice 器件仿真模型参数文件和在寄生参数提取过程中生成的网表文件 HLFF.spf，就可以进行标准单元 HLFF 的时序信息特征化实现了。

8.1.3 标准单元时序信息提取

在命令终端输入命令 slc 启动 SignalStorm Library Characterizer 工具后进入界面，如图 8-2 所示。

```
SignalStorm - Library Characterization - Version v05.20-p010_1 - linux_x86 32-bit (11/27/2005 22:35:20)
-----
[2000-2005] Cadence Design Systems, Inc. All rights reserved.
This work may not be copied, modified, re-published, uploaded, executed, or
distributed in any way, in any medium, whether in whole or in part, without
prior written permission from Cadence Design Systems, Inc.
set_var MANPATH = /CAD/soft/cadence/slc/share/slc/manpages
slc>
```

图 8-2 SLC 界面

启动 SignalStorm Library Characterizer 后，需要建立一个工作数据库，这样当前所有的工作数据都在该数据库中保存。本单元特征化实战过程是基于典型情况下，定义名为 TT 的设计数据库，用于对时序单元 HLFF 的时序特征化实现。在 SignalStorm Library Characterizer 命令输入端输入命令 db_open TT，打开一个设计数据库，其结果如图 8-3 所示。

如果已经存在名为 TT 的设计数据库，那么 SignalStorm 工具直接打开该设计数据库；如果不存在名为 TT 的设计数据库，那么 SignalStorm 工具会自动建立名为 TT 的设计数据库。

```
slc> db_open TT
Database : TT is now opened
slc>
```

图 8-3 打开设计数据库

用于单元特征化实现的设计库建立后，需要设置特征化流程的几个主要的环境变量，环境变量内容如表 8-1 所示。

表 8-1 环境变量内容

| 环境变量用途 | 环境变量命令 |
|-------------|--|
| 定义日志文件 | set_var SG_SIM_LOG "TT.log" |
| 定义仿真结果输出文件 | set_var SG_SIM_WORK "TT.work" |
| 定义所使用的仿真器工具 | set_var SG_SIM_TYPE hspice set_var SG_SIM_NAME hspice |
| 定义是否进行功耗计算 | set_var SG_LEAKAGE_VECTOR 1 |

通过定义日志文件可以随时了解时序特征化流程中的细节信息,通过定义仿真结果输出文件可以得到工具特征化过程中所使用的仿真数据和结果。定义所使用的仿真工具用来指导工具调用正确的可用电路仿真软件进行时序特征化仿真,这里定义使用 hspice 电路仿真器。定义是否进行功耗计算,由于功耗信息在当前先进工艺下变得越来越重要,因此这里定义进行功耗计算。

在 SignalStorm Library Characterizer 命令输入端输入如表 8-1 所示的环境变量设置命令,其结果如图 8-4 所示。

环境变量设置完成后,接下来是导入基本设计数据:时序单元 HLFF 的寄生参数网表和模拟所需的器件模型参数文件。通过输入命令 db_install -model typical.hspice -subckt HLFF.spf -corner "tt" 导入基本设计数据,其结果如图 8-5 所示。

以上信息显示时序单元 HLFF 的数据已经成功导入,同时 SignalStorm 工具基于导入的器件模型文件也识别出组成时序单元 HLFF 的最小组成器件:名为 P18 的 PMOS 晶体管和名为 N18 的 NMOS 晶体管。

基本设计数据导入完成后,下一步是对时序单元 HLFF 进行单元功能识别,并自动产生仿真模拟向量。通过输入命令 db_gsim 进行单元功能识别并自动产生仿真模拟向量,其结果如图 8-6 所示。

以上显示 SignalStorm 工具成功识别了时序单元 HLFF 的逻辑功能,并产生了对应的用于进行时序仿真和功耗仿真的 12 组仿真向量,同时还显示了单元网表及特性文件定义文件地址和单元结构归纳信息。

HLFF 单元逻辑功能识别成功后,就需要导入库设计规范文件中所设置的参数,以指导 SignalStorm 工具基于正确的设计参数来完成模拟仿真。通过输入命令 db_setup -s setup.ss -p typical 导入库设计规范文件,其结果如图 8-7 所示。

成功导入库设计规范文件后,模拟仿真向量和仿真环境参数设置都已经准备就绪,接下来就是调用 hspice 电路仿真器进行时序单元 HLFF 的电路模拟仿真了。通过输入命令 db_spice -p typical 进行电路模拟仿真,其结果如图 8-8 所示。

```
slc> set_var SG_SIM_LOG "TT.log"
set_var SG_SIM_LOG = TT.log
TT.log
slc> set_var SG_SIM_WORK "TT.work"
set_var SG_SIM_WORK = TT.work
TT.work
slc> set_var SG_LEAKAGE_VECTOR 1
set_var SG_LEAKAGE_VECTOR = 1
1
slc> set_var SG_SIM_TYPE hspice
set_var SG_SIM_TYPE = hspice
hspice
slc> set_var SG_SIM_NAME hspice
set_var SG_SIM_NAME = hspice
hspice
slc>
```

图 8-4 环境变量设置结果

```
Reading SUBCKT:HLFF
Expanding Parameter of SUBCKT:HLFF
Writing : TT.ipdb/HLFF.design
Writing : TT.ipdb/N18.device
Writing : TT.ipdb/P18.device
slc>
```

图 8-5 导入基本设计数据



图 8-6 产生仿真模拟向量

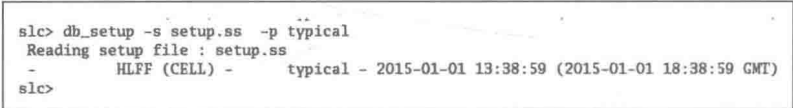


图 8-7 导入库设计规范文件

基于 12 组模拟仿真向量的电路仿真都成功完成了，这样时序单元 HLFF 的时序特征化主要阶段就基本完成了，最后是特征化结果输出。SignalStorm 工具输出最终结果之前需要先输出用于过渡的中间结果文件，以 .alf 文件保存。通过输入命令 db_output-alf HLFF.alf-p typical 输出中间结果文件，其结果如图 8-9 所示。

中间结果文件输出完成后，就可以基于中间结果文件输出真正的特征化结果了，最主要的特征化结果为时序信息文件 (.lib) 和 verilog/VHDL 逻辑网表文件。首先输出时序信息文件，除了需要读入中间结果文件，还需要读入在数据准备阶段完成的单元属性定义文件。通过输入命令 alf2lib-alf HLFF.alf-lib HLFF.lib-def property.def 输出时序信息文件，其结果如图 8-10 所示。

时序信息文件输出完成后继续输出 Verilog/VHDL 逻辑网表文件。通过输入命令 alf2vhdl-alf HLFF.alf-vhdl HLFF.vhdl 输出时序单元的 VHDL 逻辑网表，其结果如图 8-11 所示。

```

=====
HLFF      typical      20      20
-_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
```

图 8-8 电路模拟仿真结果

```
slc> db_output -alf HLFF.alf -p typical
HLFF          typical  2015-01-01 14:09:22 (2015-01-01 19:09:22 GMT) 20 (100%)
slc>
```

图 8-9 输出中间结果文件

```
SignalStorm - LC/DC - Version v05.20-p010_1 32-bit (11/27/2005 22:22:44)

[2000-2005] Cadence Design Systems, Inc. All rights reserved.
This work may not be copied, modified, re-published, uploaded, executed, or
distributed in any way, in any medium, whether in whole or in part, without
prior written permission from Cadence Design Systems, Inc.
Convert: "HLFF.alf" (ALF) ==> "HLFF.lib" (Liberty)

Library: TT
-----
CELL HLFF:
  now reading
  now converting
  ***** successful *****
-----
Total : 1 cells ( successful : 1 failed : 0 )

normal termination ( program name : alf2lib error : 0 warning : 0 )
-----
slc>
```

图 8-10 输出时序信息文件


```

slc> alf2vhd1 -alf HLFF.alf -vhd1 HLFF.vhd1
-----
SignalStorm - LC/DC - Version v05.20-p010_1 32-bit (11/27/2005 22:22:44)
-----
[2000-2005] Cadence Design Systems, Inc. All rights reserved.
This work may not be copied, modified, re-published, uploaded, executed, or
distributed in any way, in any medium, whether in whole or in part, without
prior written permission from Cadence Design Systems, Inc.
Convert: "HLFF.alf" (ALF) ==> "HLFF.vhd1" (VHDL)

Library: TT
-----
CELL HLFF:
  now reading
  now converting
  ***** successful *****
-----
Total : 1 cells ( successful : 1 failed : 0 )
-----
normal termination ( program name : alf2vhd1 error : 0 warning : 0 )
-----
slc>

```

图 8-11 输出 VHDL 逻辑网表

通过输入命令 `alf2veri-alf HLFF.alf-verilog HLFF.v` 来分别输出时序单元的 verilog 逻辑网表, 其结果如图 8-12 所示。

```

slc> alf2veri -alf HLFF.alf -verilog HLFF.v
-----
SignalStorm - LC/DC - Version v05.20-p010_1 32-bit (11/27/2005 22:22:44)
-----
[2000-2005] Cadence Design Systems, Inc. All rights reserved.
This work may not be copied, modified, re-published, uploaded, executed, or
distributed in any way, in any medium, whether in whole or in part, without
prior written permission from Cadence Design Systems, Inc.
Convert: "HLFF.alf" (ALF) ==> "HLFF.v" (Verilog)

Library: TT
-----
CELL HLFF:
  now reading
  now converting
  ***** successful *****
-----
Total : 1 cells ( successful : 1 failed : 0 )
-----
normal termination ( program name : alf2veri error : 0 warning : 0 )
-----
slc>

```

图 8-12 输出 verilog 逻辑网表

时序信息文件 (.lib) 和 verilog/VHDL 逻辑网表文件输出完成后, 时序特征化过程就基本完成了。通过输入命令 `db_close` 关闭设计数据库, 如图 8-13 所示。

正常关闭设计数据库的目的是保持设计数据库的完整性, 在以后的设计过程中可以通过命令 `db_open` 重新打开该设计数据库。

```

slc> db_close
Database : TT is closed

```

图 8-13 关闭设计数据库

8.2 SiliconSmart 工具的使用流程简介

业界使用的特性参数提取工具中还包括被 Synopsys 公司收购的 Magma 公司的 SiliconSmart,

由于时序信息特征化工作非常小众,该工具至今还有部分业界设计人员使用,因此本节简单介绍下 SiliconSmart 的基本工作流程。

在使用 SiliconSmart 工具对单元库进行特性提取时,主要有两种方法,一种是在有 lib 模板情况下,另一种是在没有模板情况下。一般情况下,我们的单元都是基于流片厂家来设计的,所以一般都会有厂家提供的单元库 lib 作为模板,所以我们是在有 lib 模板的情况下进行单元库设计的。

下面对 SiliconSmart 工具的基本使用流程说明如下, SiliconSmart 工具的使用流程分为 8 个步骤。

1) create 文件名。

创建执行文件夹,程序把要生成的文件都创建到该文件夹下。

2) 配置 configure.Tcl 脚本内容。

create 文件后就生成了 configure.Tcl,为了让工具合理的对单元库进行模拟,需要在这一步做关键的指导配置,主要包括模拟软件的选择(hspice、finestim 等)、模拟模型的指定、环境温度等变量设置、VSS 和 VDD 的设置、模拟方式中参数的设置(largest_load 等)。

3) set_location 文件名(同上一致)。

这一步要告诉软件运行下的文件夹,这一步是必需的,而且只要改动了 configure.Tcl 脚本就要调用这个命令,以更新设置。

4) import-liberty 地址 /*.lib all 或者其中的部分单元。

导入参考 lib 文件,使软件提取出单元的功能等信息。

5) 加入网表到 nelist 文件夹下。

如果我们使用的网表格式不是 .cir 那么需要修改 Control 文件夹下的 .inst 文件中的格式,如果是一致的就不需要修改。 .inst 文件在 control 文件夹下,导入 lib 信息后才能生成。加入网表 *.cir 或者 *.spf 等到 netlist 文件夹下。

6) configure 参数。

网表输入和配置都完成后就可以让软件进行配置设置和网表的读入工作了,为后面的特性提取做好最后的准备。这一步是告诉软件将要进行哪些方面的特性提取工作,比如信号完整性、功耗、CCS 模型等。

7) char 单元名或全部。

对单元进行模拟仿真特性提取工作。

8) model 单元名或全部。

对单元进行建模并输出结果文件,即 lib。

8.3 时序信息提取内容

时序信息提取内容如下:

```

library(TT) {

    delay_model : table_lookup;
    in_place_swap_mode : match_footprint;

    /* unit attributes */
    time_unit : "1ns";
    voltage_unit : "1V";
    current_unit : "1uA";
    pulling_resistance_unit : "1kohm";
    leakage_power_unit : "1nW";
    capacitive_load_unit (1,pf);

    slew_upper_threshold_pct_rise : 70;
    slew_lower_threshold_pct_rise : 30;
    slew_upper_threshold_pct_fall : 70;
    slew_lower_threshold_pct_fall : 30;
    input_threshold_pct_rise : 50;
    input_threshold_pct_fall : 50;
    output_threshold_pct_rise : 50;
    output_threshold_pct_fall : 50;
    nom_process : 1;
    nom_voltage : 1.2;
    nom_temperature : 25;
    operating_conditions ( typical ) {
        process : 1;
        voltage : 1.2;
        temperature : 25;
    }
    default_operating_conditions : typical;

    lu_table_template(delay_template_7x7) {
        variable_1 : total_output_net_capacitance;
        variable_2 : input_net_transition;
        index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0, 1005.0, 1006.0");
        index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0, 1005.0, 1006.0");
    }

    power_lut_template(energy_template_7x7) {
        variable_1 : total_output_net_capacitance;
        variable_2 : input_transition_time;
        index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0, 1005.0, 1006.0");
        index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0, 1005.0, 1006.0");
    }

    lu_table_template(hold_template_3x3) {
        variable_1 : related_pin_transition;
        variable_2 : constrained_pin_transition;
        index_1 ("1000.0, 1001.0, 1002.0");
        index_2 ("1000.0, 1001.0, 1002.0");
    }

    power_lut_template(passive_energy_template_7x1) {

```

```

        variable_1 : input_transition_time;
        index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0, 1005.0, 1006.0");
    }
    lu_table_template(setup_template_3x3) {
        variable_1 : related_pin_transition;
        variable_2 : constrained_pin_transition;
        index_1 ("1000.0, 1001.0, 1002.0");
        index_2 ("1000.0, 1001.0, 1002.0");
    }
    lu_table_template(width_template_7x1) {
        variable_1 : related_pin_transition;
        index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0, 1005.0, 1006.0");
    }

/* ----- *
 * Design : HLFF *
 * ----- */
cell (HLFF) {
    area : 0.3;
    cell_leakage_power : 0.0920476;
    ff (P0002,P0003) {
        next_state : "D";
        clocked_on : "CLK";
    }
    pin(CLK) {
        direction : input;
        capacitance : 0.00230994;
        rise_capacitance : 0.00230994;
        fall_capacitance : 0.00222246;
        clock : true;
        internal_power() {
            rise_power(passive_energy_template_7x1) {
                index_1 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
                values ("0.0109782, 0.0128619, 0.0104243, 0.010911, 0.0105612, 0.010928,
0.0115386");
            }
            fall_power(passive_energy_template_7x1) {
                index_1 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
                values ("0.0174204, 0.0177603, 0.0169089, 0.018011, 0.0169138, 0.0178689,
0.0186739");
            }
        }
    }
    timing() {
        related_pin : "CLK";
        timing_type : min_pulse_width;
        rise_constraint(width_template_7x1) {
            index_1 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
            values ("0.424752, 0.441968, 0.518507, 0.579829, 0.60206, 0.628954,
0.662295");
        }
    }
}

```

```

        fall_constraint(width_template_7x1) {
            index_1 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
            values ("0.597811, 0.613125, 0.693766, 0.853993, 1.04973, 1.25838,
1.45267");
        }
    }
}
pin(D) {
    direction : input;
    capacitance : 0.00132032;
    rise_capacitance : 0.00131102;
    fall_capacitance : 0.00132032;
    internal_power() {
        rise_power(passive_energy_template_7x1) {
            index_1 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
            values ("0.0094412, 0.0080306, 0.0086485, 0.0085174, 0.00864, 0.0087027,
0.0088874");
        }
        fall_power(passive_energy_template_7x1) {
            index_1 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
            values ("0.0067022, 0.0083312, 0.0083731, 0.0082025, 0.0090153, 0.0090591,
0.0099489");
        }
    }
}
timing() {
    related_pin : "CLK";
    timing_type : hold_rising;
    rise_constraint(hold_template_3x3) {
        index_1 ("0.015, 0.75, 1.1");
        index_2 ("0.015, 0.75, 1.5");
        values ( \
            "-0.0545697, -0.16523, -0.10994", \
            "-0.0383799, -0.12158, -0.07288", \
            "-0.0594999, -0.1394, -0.10828");
    }
    fall_constraint(hold_template_3x3) {
        index_1 ("0.015, 0.75, 1.1");
        index_2 ("0.015, 0.75, 1.5");
        values ( \
            "-0.15564, -0.62886, -1.0207", \
            "0.0286398, -0.41272, -0.78479", \
            "0.0547598, -0.4613, -0.84875");
    }
}
timing() {
    related_pin : "CLK";
    timing_type : setup_rising;
    rise_constraint(setup_template_3x3) {
        index_1 ("0.015, 0.75, 1.1");
        index_2 ("0.015, 0.75, 1.5");
    }
}

```

```

        values ( \
            "0.0776001, 0.21025, 0.18135", \
            "0.0603499, 0.1523, 0.1333", \
            "0.0836504, 0.1636, 0.1456");
    }
    fall_constraint(setup_template_3x3) {
        index_1 ("0.015, 0.75, 1.1");
        index_2 ("0.015, 0.75, 1.5");
        values ( \
            "0.3116, 0.78045, 1.17012", \
            "0.0933497, 0.6094, 1.0199", \
            "0.0946505, 0.5657, 6.72754");
    }
}

pin(Q) {
    direction : output;
    capacitance : 0;
    rise_capacitance : 0;
    fall_capacitance : 0;
    max_capacitance : 0.366119;
    function : "P0002";
    timing() {
        related_pin : "CLK";
        timing_sense : non_unate;
        timing_type : rising_edge;
        cell_rise(delay_template_7x7) {
            index_1 ("0.00035, 0.021, 0.0385, 0.084, 0.147, 0.231, 0.3115");
            index_2 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
            values ( \
                "0.378019, 0.394253, 0.472536, 0.531699, 0.540662, 0.569607, 0.593524", \
                "0.550023, 0.566029, 0.641516, 0.701515, 0.722866, 0.749977, 0.78023", \
                "0.685794, 0.701608, 0.77695, 0.836702, 0.857887, 0.885102, 0.91433", \
                "1.03685, 1.05102, 1.12642, 1.19041, 1.20926, 1.23519, 1.26442", \
                "1.52142, 1.53666, 1.61142, 1.68064, 1.69167, 1.71966, 1.74939", \
                "2.16518, 2.18412, 2.25676, 2.3222, 2.33712, 2.37314, 2.39355", \
                "2.78368, 2.82137, 2.87602, 2.94288, 2.9688, 2.98295, 3.0182");
        }
        rise_transition(delay_template_7x7) {
            index_1 ("0.00035, 0.021, 0.0385, 0.084, 0.147, 0.231, 0.3115");
            index_2 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
            values ( \
                "0.0334123, 0.0367971, 0.0384231, 0.0406605, 0.0670733, 0.0684763,
0.0770556", \
                "0.168796, 0.168987, 0.169357, 0.169746, 0.167049, 0.167607, 0.168287", \
                "0.292857, 0.29172, 0.291384, 0.29386, 0.289413, 0.290211, 0.289959", \
                "0.606915, 0.612148, 0.610866, 0.654017, 0.625421, 0.610375, 0.608321", \
                "1.0467, 1.06571, 1.0476, 1.0809, 1.06577, 1.07322, 1.04714", \
                "1.63962, 1.72277, 1.63947, 1.66841, 1.66686, 1.67691, 1.6886", \
                "2.20657, 2.2384, 2.23838, 2.25096, 2.24571, 2.22308, 2.24781");
        }
    }
}

```

```

    }
    cell_fall(delay_template_7x7) {
        index_1 ("0.00035, 0.021, 0.0385, 0.084, 0.147, 0.231, 0.3115");
        index_2 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
        values ( \
            "0.303364, 0.320475, 0.38805, 0.449277, 0.464751, 0.496702,
0.519342", \
            "0.446846, 0.463882, 0.532196, 0.593343, 0.617996, 0.644126,
0.679146", \
            "0.525029, 0.542055, 0.610788, 0.672544, 0.696716, 0.723495,
0.757309", \
            "0.708173, 0.725133, 0.793648, 0.85574, 0.879363, 0.906135,
0.939977", \
            "0.955524, 0.972515, 1.04083, 1.10204, 1.12674, 1.15275, 1.18707", \
            "1.28318, 1.30018, 1.36862, 1.43313, 1.45409, 1.48118, 1.51462", \
            "1.60119, 1.61803, 1.68703, 1.74581, 1.77555, 1.79602, 1.83011");
    }
    fall_transition(delay_template_7x7) {
        index_1 ("0.00035, 0.021, 0.0385, 0.084, 0.147, 0.231, 0.3115");
        index_2 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
        values ( \
            "0.0428914, 0.0426117, 0.0433741, 0.052253, 0.107557, 0.0961359,
0.12546", \
            "0.106441, 0.106571, 0.106955, 0.109959, 0.11504, 0.113335,
0.119629", \
            "0.155708, 0.155692, 0.156259, 0.160453, 0.158474, 0.15817,
0.161276", \
            "0.296845, 0.297185, 0.298065, 0.301735, 0.297527, 0.296924,
0.297158", \
            "0.509866, 0.509125, 0.509592, 0.507049, 0.509057, 0.503574,
0.50515", \
            "0.794437, 0.793412, 0.795471, 0.790461, 0.814042, 0.788627,
0.782852", \
            "1.0873, 1.08707, 1.08652, 1.07249, 1.0831, 1.05175, 1.05722");
    }
}
internal_power() {
    related_pin : "CLK";
    rise_power(energy_template_7x7) {
        index_1 ("0.00035, 0.021, 0.0385, 0.084, 0.147, 0.231, 0.3115");
        index_2 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
        values ( \
            "0.0204825, 0.0204014, 0.0205855, 0.0211241, 0.0205141, 0.0206999,
0.0214162", \
            "0.0209929, 0.0209382, 0.0209113, 0.0214219, 0.0208917, 0.0210328,
0.0216129", \
            "0.0211301, 0.0210662, 0.0210282, 0.0215853, 0.0210448, 0.0211756,
0.0217313", \
            "0.0212906, 0.0211969, 0.021215, 0.0216989, 0.0212259, 0.0213593,
0.0219005", \

```

```

0.0220238", \
0.0221108", \
0.0221572");
    }
    fall_power(energy_template_7x7) {
        index_1 ("0.00035, 0.021, 0.0385, 0.084, 0.147, 0.231, 0.3115");
        index_2 ("0.015, 0.05, 0.2, 0.45, 0.75, 1.1, 1.5");
        values ( \
            "0.0234519, 0.0233499, 0.0226096, 0.0237111, 0.0233474, 0.0230098,
            "0.0236893, 0.0236222, 0.0228338, 0.0237771, 0.0236107, 0.0232915,
            "0.0237772, 0.0237939, 0.0228683, 0.0238855, 0.0235524, 0.0231881,
            "0.0237646, 0.0236435, 0.0228886, 0.0239751, 0.0237078, 0.0232905,
            "0.024355, 0.0243118, 0.0232622, 0.0240321, 0.0237634, 0.0233773,
            "0.0239768, 0.023871, 0.0230579, 0.0240619, 0.0237778, 0.0233919,
            "0.0240001, 0.0238927, 0.0230841, 0.0240691, 0.0237999, 0.0234233,
            0.0243145", \
            0.0246787", \
            0.0245296", \
            0.0245187", \
            0.0245919", \
            0.0246648", \
            0.0246426");
        }
    }
}
}
}

```


第 9 章

静态时序分析实战 (ETS 篇)

本章将基于 OpenSparcT1 里浮点计算单元部件 (floating-point processor unit) 来讲解静态时序分析过程。关于 OpenSparcT1 的详细代码理论知识,读者可以参阅其他以代码理论基础为主的相关书籍,本章是学习如何在时序上进行分析。

本章时序分析过程所使用的 EDA 工具为 Cadence 公司的 Encounter Timing System 工具,业界简称 ETS。

9.1 静态时序分析的基本流程

布局布线完成以后,需要对最终设计数据进行静态时序分析,来保证设计能工作在稳定的工作频率下。静态时序分析流程主要分为以下 5 步进行介绍,如图 9-1 所示。

1. 静态时序分析数据准备

静态时序分析之前需要准备好用于设计的数据,比如门级网表、时序约束文件和时序库文件等。

2. 建立时间分析

建立时间分析是分析在时钟捕获信号有效沿到来前,数据信号必须提前达到稳定状态时间是否满足要求。时序路径分为 4 种类型:触发器到触发器、触发器到输出端、输入端到触发器和输入端到输出端。因此建立时间分析中需要分别对这 4 种类型的时序路径进行分析。

3. 保持时间分析

保持时间分析是分析为保证时序单元对数据读取正确,数据在时钟有效沿到来之后,仍需要保持稳定的时间是否满足要求。与建立时间分析一样,保持时间分析中也需要分别对 4 种类型的时序路径进行分析。

4. 设计规则分析

时序路径除了满足建立时间和保持时间的时序要求以外,也需要满足时序设计规则的要

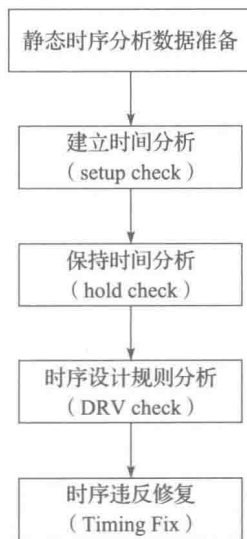


图 9-1 静态时序分析流程

求,因此还要对进行时序设计规则检查(DRV)。时序设计规则主要包括3个方面:最大转换时间、最大电容负载、最大扇出。

5. 时序违反修复

静态时序分析不仅能够分析出不满足时序约束的时序路径,也可以根据时序分析报告进行时序优化来修复存在时序违反的时序路径。

9.2 建立静态时序分析的工作环境

在命令终端输入 ets 命令启动 cadence ETS,进入静态时序分析工作界面,如图 9-2 所示。



图 9-2 ETS 工作界面

在工作界面选择栏中选择 Setup 选项,进入设计数据导入设置界面,如图 9-3 所示。

设计数据导入设置界面中有两个分类设置框需要设置: Library 设置框、Design 设置框。

在 Library 设置框中输入正确的时序库的地址,如图 9-4 所示。

通常情况时序库根据实际的工作环境分为最快的、最慢的、正常的这3种库,工具根据时序分析算法的需要来调用不同的时序库。由于静态时序分析数据来自布局布线设计的结果,因此时序库与布局布线的时序库设置保持一致。

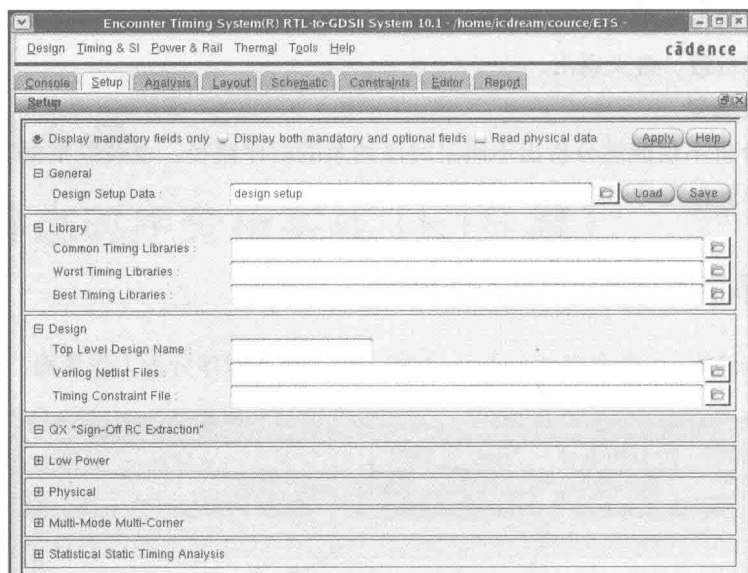


图 9-3 数据导入设置界面

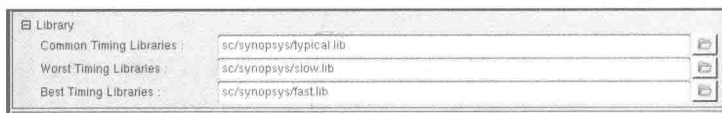


图 9-4 Library 设置框

在 Design 设置框中填入正确的逻辑设计网表、时序约束文件，如图 9-5 所示。

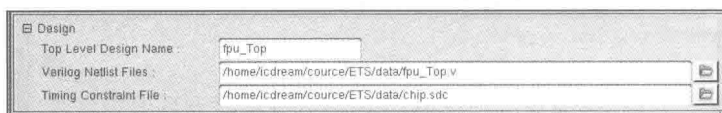


图 9-5 Design 设置框

逻辑设计网表为布局布线设计完成后输出结果，时序约束文件与布局布线设计的设置保持一致。因为实战流程是基于 OpenSparcT1 里浮点计算单元部件，示例中逻辑网表最顶层模块名为 fpu_Top。

Library 设置框和 Design 设置框设置完成后，单击设计数据导入设置界面中右上角的 Apply 按钮，这样建立基本的静态时序分析工作环境就完成了。数据导入后，可以在工作界面选择栏中选择不同的工作信息显示选项，单击 Console 选项，进入设计数据导入后的控制台信息显示界面，如图 9-6 所示。

在工作界面选择栏中单击 Schematic 选项，进入设计数据导入后的原理图显示界面，如图 9-7 所示。

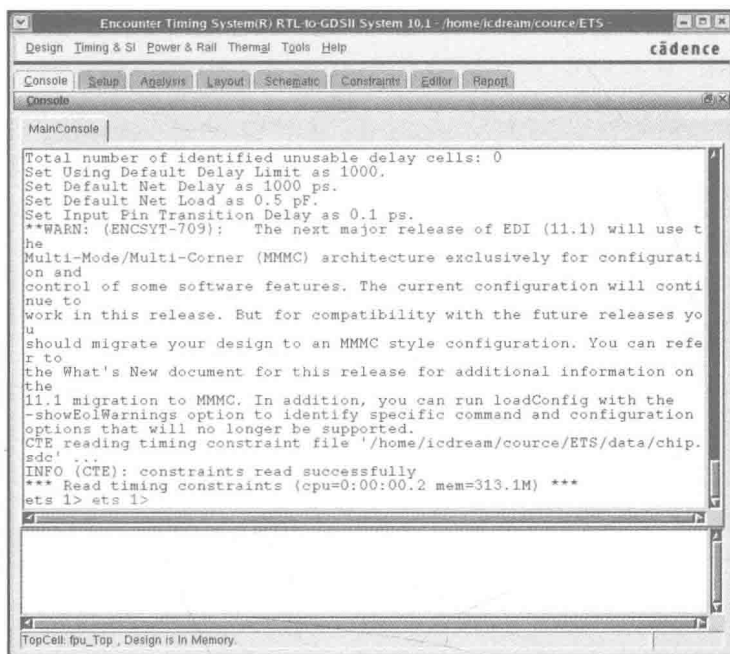


图 9-6 控制台信息显示界面

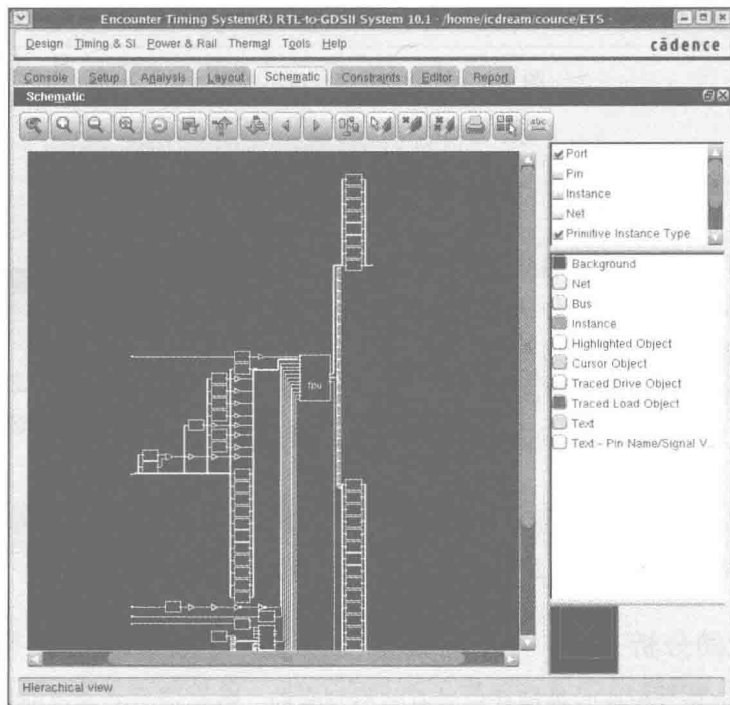


图 9-7 原理图显示界面

在工作界面选择栏中单击 Constraints 选项，进入设计数据导入后时序约束信息显示界面，如图 9-8 所示。

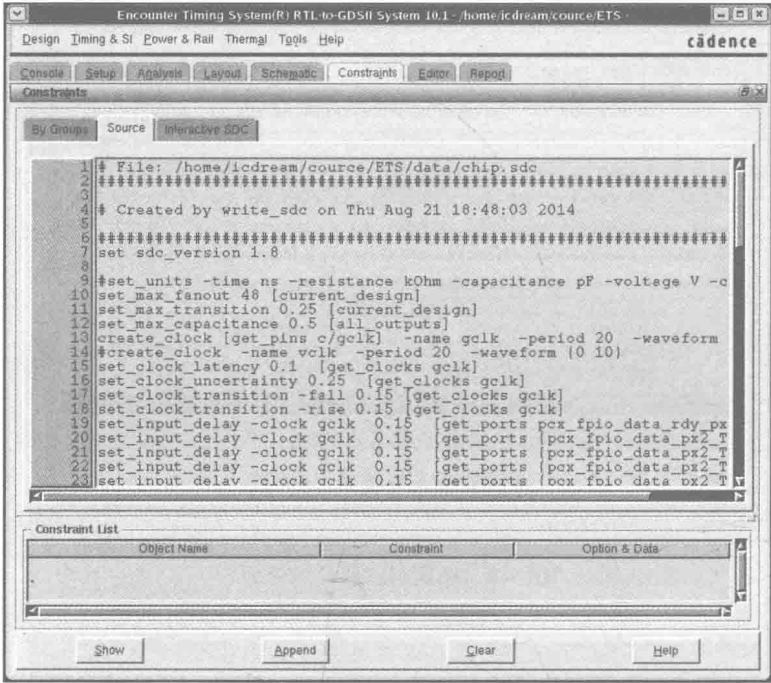


图 9-8 时序约束信息显示界面

9.3 静态时序分析实现

完成静态时序分析工作环境建立后，在进行静态时序分析之前，为了得到精确的时序分析结果，需要先读入寄生参数信息，而寄生参数信息文件为布局布线设计完成后输出结果。在菜单栏里单击 Design → Read SPEF，如图 9-9 所示。



图 9-9 选择读入寄生参数设置

在寄生参数文件设置界面下选择正确的寄生参数信息文件，如图 9-10 所示。

单击 Open 按钮后就完成了寄生参数信息文件的导入工作，其结果如图 9-11 所示。完成寄生参数信息读取后，就可以对设计进行静态时序分析工作了。

9.3.1 建立时间分析

建立时间分析是分析在时钟捕获信号有效沿到来前，数据信号必须提前达到稳定状态时间是否满足要求。前面章节讲到，时序路径分为 4 种类型：触发器到触发器、触发器到输出

端、输入端到触发器和输入端到输出端。因此建立时间分析中需要分别对这4种类型的时序路径进行分析。



图 9-10 选择正确的寄生参数信息文件

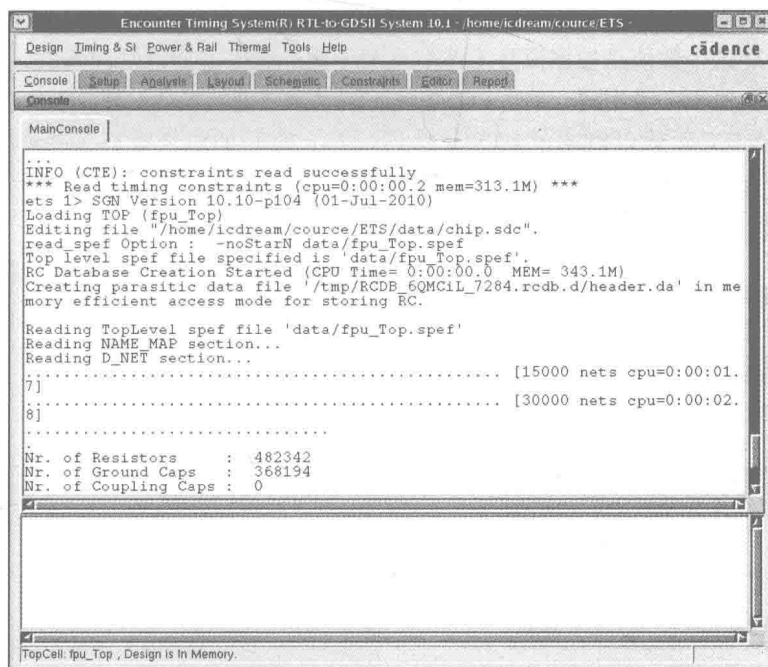


图 9-11 寄生参数信息文件导入完成

1. 建立时间分析模式设置

在 ETS 中进行建立时间分析之前,需要定义静态时序分析模式,在菜单栏里单击 Timing&SI → Specify Timing Analysis Mode,如图 9-12 所示。

静态时序分析模式设置界面有两个分类设置栏：Basic 设置栏和 Advanced 设置栏。这里主要是对 Basic 设置栏进行设置，Advanced 设置栏保持默认值。由于 Basic 栏是默认显示的，其设置界面如图 9-13 所示。

在 Timing Check 设置框中，选择 Setup 选项来定义进行建立时间分析。在 Timing Mode 设置框中选择 Best and Worst Case 选项，来定义静态时序分析方法基于 WC_BC 模式。为了减少在计算时钟网络公共路径延时导致的悲观影响，选中 CPPR 复选项，设计中如果存在门控时钟设计，那么也需要选中 Clock Gating Check 复选项，最后单击 OK 按钮完成静态时序分析模式的设置。

由于设置了 CPPR 功能，那么还需要设置时序减免 (timing derate)。在菜单栏里单击 Timing&SI → Set Timing Derate，如图 9-14 所示。

进入时序减免设置界面，如图 9-15 所示。

通过设置时序减免值来对时序分析结果进行裕量设置，这里设置 Early 选项中的值为 0.94，使得时序分析结果更加合理，0.94 这个数值是根据流片工艺节点和工程经验等方面评估而来的，不同的工艺和工程中，时序减免值不一定相同。单击 OK 按钮完成时序减免的参数设置。

时序减免设置完后就可以通过 ETS 软件进行建立时间的时序计算，并产生时序分析结果。在菜单栏里单击 Timing&SI → Timing Analysis，如图 9-16 所示。

进入时序分析设置界面，如图 9-17 所示。

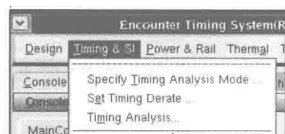


图 9-12 选择时序分析模式设置

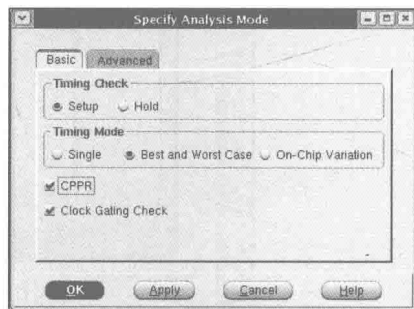


图 9-13 静态时序分析模式设置界面

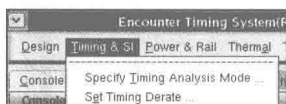


图 9-14 选择时序减免

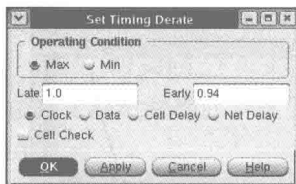


图 9-15 时序减免设置界面

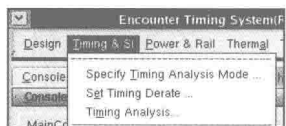


图 9-16 选择时序分析设置

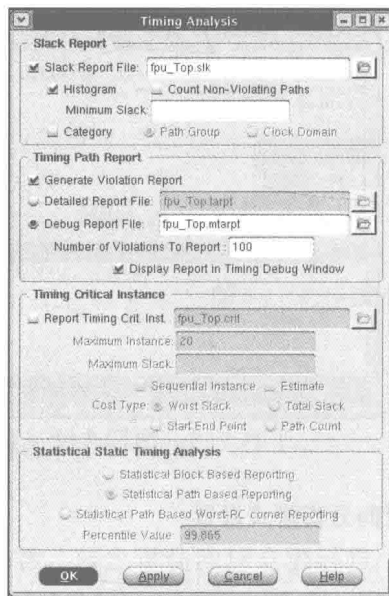


图 9-17 时序分析设置界面

在 Timing Path Report 设置栏中选中 Generate Violation Report 和 Debug Report File 这两个选项。在 Debug Report File 文本框中设置正确的时序报告文件名称,并定义时序报告中所记录的时序路径数量,这里设置为 100。这里时序报告文件名称为 fpu_Top.mtarpt,该报告文件提供给 ETS 工具内嵌的图形化时序分析器使用,这样可以帮助设计者进行基于图形界面的时序分析工作。最后单击 OK 按钮,ETS 软件就开始进行建立时间的时序计算,并产生时序分析结果。

2. 图形化时序分析器

ETS 软件完成建立时间的时序计算以后,在工作界面选择栏中单击 Analysis 选项,进入图形化时序分析界面,如图 9-18 所示。

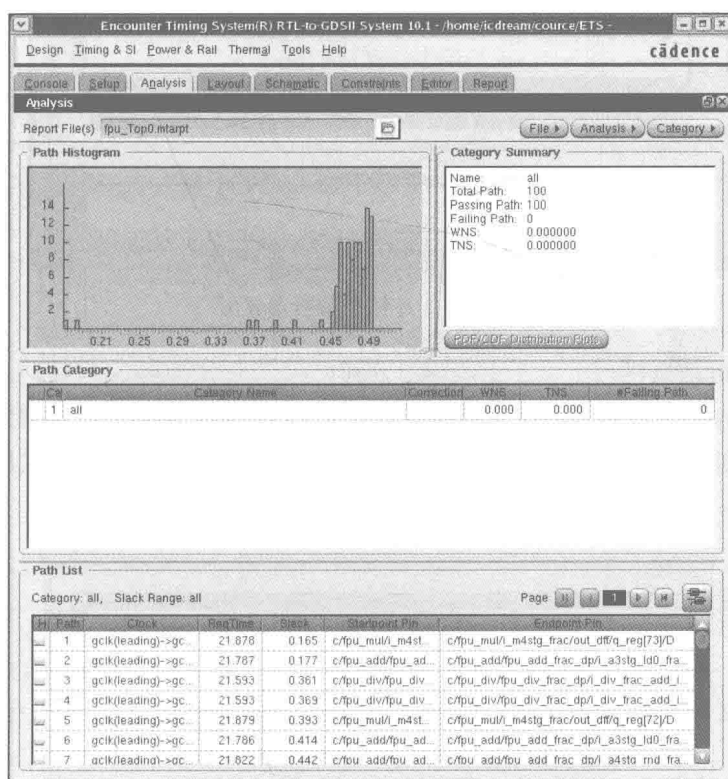


图 9-18 图形化时序分析界面

该图形化时序分析界面为 ETS 内嵌的图形化时序分析器界面,在 Path List 中双击其中一条时序路径,进入时序路径分析器界面,如图 9-19 所示。

在时序路径分析器的上半部分显示了该路径的类型、时序路径起点、时序路径终点、时钟延时时和时钟偏斜值等全局信息。在 Slack calculation 信息栏中以图形化的表达方式显示了时序路径的全局信息。

在时序路径分析器界面下方还有以下 7 个分类信息栏。

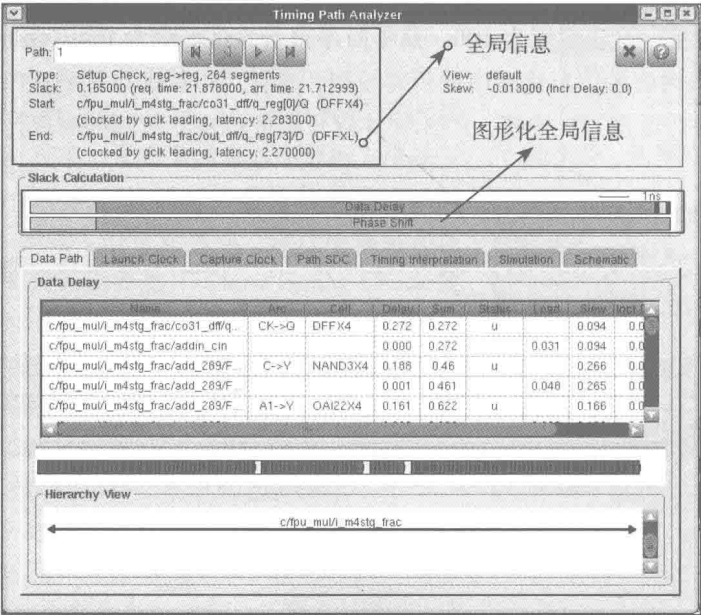


图 9-19 时序路径分析器界面

(1) Data Path 信息栏

在 Data Delay 信息栏中主要以表格的形式显示了该条时序路径中的详细延时信息，比如单元名称、单元延时、线延时等信息。同时也在下方通过图形化的表达方式显示了该条时序路径的全局信息，如图 9-20 所示。

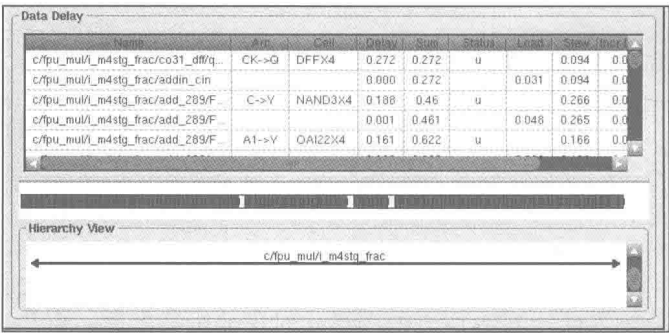


图 9-20 Data Path 信息栏

(2) Launch Clock 信息栏

Launch Clock 信息栏中主要以表格的形式显示了发射时钟路径的详细延时信息，同时也

在下方通过图形化的表达方式显示了该条时序路径的全局信息，如图 9-21 所示。

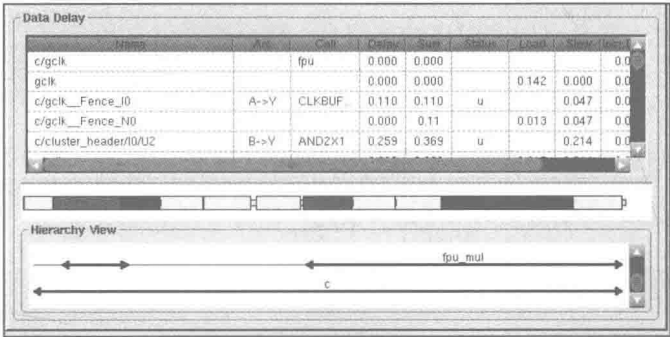


图 9-21 Launch Clock 信息栏

(3) Capture Clock

Capture Clock 信息栏中主要以表格的形式显示了捕获时钟路径的详细延时信息，同时也在下方通过图形化的表达方式显示了该条时序路径的全局信息，如图 9-22 所示。

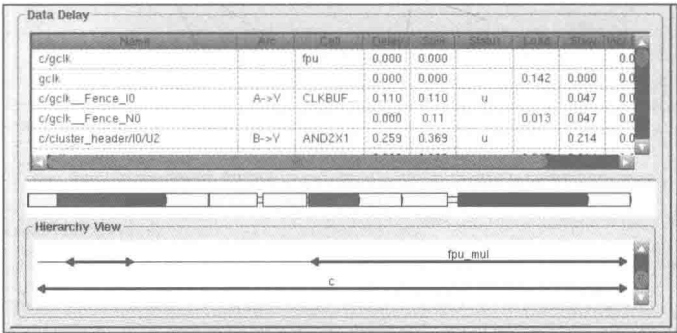


图 9-22 Capture Clock 信息栏

(4) Path SDC

Path SDC 信息栏中显示了该条时序路径相关的时序约束信息，如图 9-23 所示。



图 9-23 Path SDC 信息栏

(5) Timing Interpretation

Timing Interpretation (时序解释) 信息栏中显示了该条时序路径更详细的解释信息，比如路径结构组成信息、路径延时和时序约束信息、路径物理信息和 DRV 信息，如图 9-24 所示。

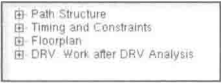


图 9-24 时序解释信息栏

(6) Simulation

Simulation (模拟) 信息栏中显示了通过该条时序路径进行 SPICE 网表级仿真而得到的更精确的时序延时信息，如图 9-25 所示。

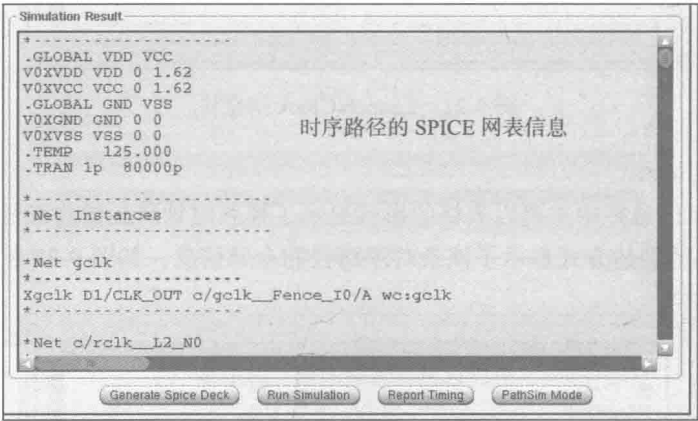


图 9-25 模拟信息栏

(7) Schematic

Schematic (原理图) 信息栏中显示了该条时序路径的电路原理图信息，如图 9-26 所示。

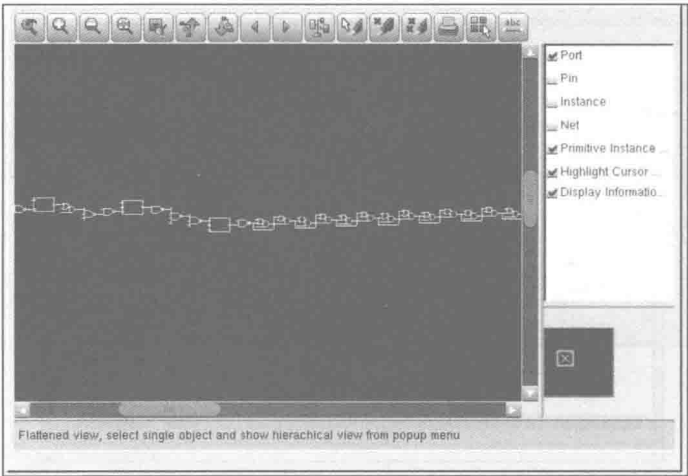


图 9-26 原理图信息栏

在时序分析器辅助下设计者可以通过图形化的方式,更直观地对每条违反的时序路径进行详细分析。不管是在建立时间分析阶段还是在保持时间分析阶段,时序分析器的使用方法都基本一致。

时序分析也可以通过基于文本形式的时序报告进行分析,由于图形化时序分析器专用时序报告文件(fpu_Top.mtarpt)的格式是按照时序分析器的读取要求而定义的,并不是用于人工阅读的时序报告,因此需要生成用于人工分析的常规时序报告。在菜单栏里单击 Timing&SI → Timing Analysis 进入时序分析设置界面,在 Timing Path Report 设置栏中选中 Generate Violation Report 和 Detailed Report File 这两个选项。在 Detailed Report File 选项中设置正确的时序报告文件名称,并定义时序报告中所记录的时序路径数量,这里设置为 50;这里时序报告文件名称为 fpu_Top.tarpt,如图 9-27 所示。

单击 OK 按钮,ETS 软件就开始进行建立时间的时序计算,并产生易于人工阅读的时序分析结果,生成的时序报告如图 9-28 所示。

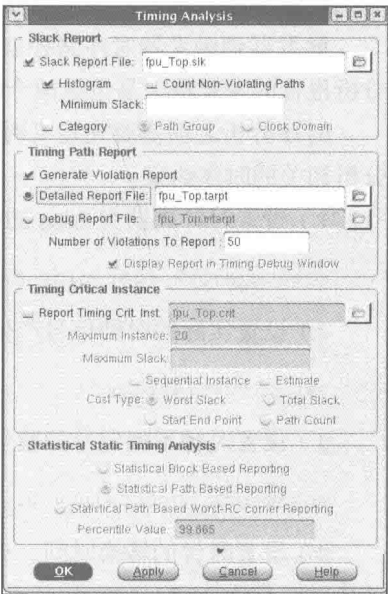


图 9-27 生成时序报告设置

```
#####
# Generated by: Cadence Encounter Timing System 10.10-p001_1
# OS: Linux x86_64(Host ID localhost.localdomain)
# Generated on: Thu Jan 1 02:30:23 2015
# Design: fpu_Top
# Command: report_timing -max_points 50 > fpu_Top.tarpt
#####
Path 1: MET Setup Check with Pin c/fpu_mul/i_m4stg_frac/out_dff/q_reg[73]/CK
Endpoint: c/fpu_mul/i_m4stg_frac/out_dff/q_reg[73]/D (v) checked with leading edge of 'gclk'
Beginpoint: c/fpu_mul/i_m4stg_frac/co3l_dff/q_reg[0]/Q (v) triggered by leading edge of 'gclk'
Other End Arrival Time 2.270
- Setup 0.142
+ Phase Shift 20.000
+ CPFR Adjustment 0.000
- Uncertainty 0.250
= Required Time 21.878
- Arrival Time 21.713
= Slack Time 0.165
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 2.283
= Beginpoint Arrival Time 2.283
```

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|---|-------------|----------|-------|--------------|---------------|
| c/fpu_mul/i_m4stg_frac/co3l_dff/q_reg[0] | CK ^ | | | 2.283 | 2.448 |
| c/fpu_mul/i_m4stg_frac/co3l_dff/q_reg[0] | CK ^ -> Q v | DFFX4 | 0.272 | 2.555 | 2.720 |
| c/fpu_mul/i_m4stg_frac/add_289/FE_RC_0_0 | C v -> Y ^ | NAND3X4 | 0.188 | 2.743 | 2.908 |
| c/fpu_mul/i_m4stg_frac/add_289/FE_RC_38_0 | A1 ^ -> Y v | OAI22X4 | 0.162 | 2.906 | 3.071 |
| c/fpu_mul/i_m4stg_frac/add_289/FE_RC_66_0 | B v -> Y ^ | NAND38X2 | 0.249 | 3.155 | 3.320 |

图 9-28 生成的时序报告

基于文本形式时序报告的时序分析将在下面详细讲解。

3. 触发器到触发器时序路径分析

触发器到触发器时序路径表示始发点和终止点为两个时序单元之间的时序路径，因此其分析报告的主要特征是记录两个时序单元之间时序路径信息。

时序分析之前需要先定义时序约束信息，以下是比较基本的与触发器到触发器时序路径分析相关的时序约束：

1) 设置时钟周期为 20ns 的 gclk 时钟，时钟 pin 脚为 c/gclk，相位占空比为 50%。

```
create_clock [get_pins c/gclk] -name gclk -period 20 -waveform {0 10}
```

2) 设置实际的时钟信号产生到时序约束中的时钟 pin 脚的延时为 0.1ns。

```
set_clock_latency 0.1 [get_clocks gclk]
```

3) 设置时钟不确定性的数值为 0.25ns，该设置主要是考虑时钟本身所带来的延时开销。

```
set_clock_uncertainty 0.25 [get_clocks gclk]
```

4) 设置时钟的转换时间，该设置用于初始化时钟 pin 脚的驱动力，这里设置上拉和下拉转换时间都为 0.15ns。

```
set_clock_transition -fall 0.15 [get_clocks gclk]
set_clock_transition -rise 0.15 [get_clocks gclk]
```

5) 设置时钟本身的传播延时为实际时钟树的延时。如果不设置该参数，时序分析软件将把时钟当作理想时钟，这样时钟间的偏斜为 0。理想时钟的设置主要用于前期设计的预估时序过程，但不符合最终的实际情况。

```
set_propagated_clock [all_clocks]
```

基于以上的时序约束信息，如下所示为 ETS 软件生成的这一条触发器到触发器时序路径的建立时间时序分析报告。

```
Endpoint: c/fpu_add/fpu_add_frac_dp/i_a3stg_ld0_frac/q_reg[63]/D (v) checked with
leading edge of 'gclk'
Beginpoint: c/fpu_add/fpu_add_ctl/i_a2stg_fracadd_frac2/q_reg[0]/Q (v) triggered by
leading edge of 'gclk'
Other End Arrival Time                2.038
- Setup                               0.131
+ Phase Shift                         20.000
+ CPPR Adjustment                     0.052
- Uncertainty                         0.250
= Required Time                       21.708
- Arrival Time                       21.610
= Slack Time                          0.099
    Clock Rise Edge                   0.000
    + Clock Network Latency (Prop)    2.263
```

| = Beginpoint Arrival Time | | 2.263 | | | |
|--|-------------|---------|-------|--------------|---------------|
| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
| c/fpu_add/fpu_add_ctl/i_a2stg_fracadd_frac2/q_reg[0] | CK ^ | | | 2.263 | 2.362 |
| c/fpu_add/fpu_add_ctl/i_a2stg_fracadd_frac2/q_reg[0] | CK ^ -> Q v | DFFX4 | 0.259 | 2.522 | 2.620 |
| c/fpu_add/fpu_add_ctl/i_a2stg_fracadd_frac2/FE_OFC5568_a2stg_fracadd_frac2 | A v -> Y v | BUF16 | 0.207 | 2.729 | 2.828 |
| c/fpu_add/fpu_add_frac_dp/U4928 | B v -> Y ^ | MXI2X2 | 0.193 | 2.922 | 3.020 |
| c/fpu_add/fpu_add_frac_dp/U4927 | B0 ^ -> Y v | OAI21X1 | 0.105 | 3.026 | 3.125 |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| (时序路径中间部分信息省略) | | | | | |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| c/fpu_add/fpu_add_frac_dp/i_a3stg_ld0_frac/U260 | A ^ -> Y v | INVX1 | 0.060 | 21.610 | 21.708 |
| c/fpu_add/fpu_add_frac_dp/i_a3stg_ld0_frac/q_reg[63] | D v | DFFXL | 0.000 | 21.610 | 21.708 |

以上所示的时序报告主要由总结报告和详细时序路径报告两个部分组成。

在总结报告中主要包含如下时序分析信息。

(1) Endpoint

Endpoint 为触发器到触发器时序路径中与捕获时钟 (Capture Clock) 相关的时序单元的相关信息, 它包括终止点单元信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_frac_dp/i_a3stg_ld0_frac/q_reg[63] 的单元, 其信号输入端口 D 端的时序弧类型信息为 falling_edge(v), 相关时钟为 gclk。

(2) Beginpoint

Beginpoint 为触发器到触发器时序路径中与发射时钟 (Launch Clock) 相关的时序单元的相关信息, 它包括始发点单元信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_ctl/i_a2stg_fracadd_frac2/q_reg[0] 的单元, 其信号输出端口 Q 端的时序弧类型信息为 falling_edge(v), 相关时钟为 gclk。

(3) Other End Arrival Time

Other End Arrival Time 为捕获时钟的自身传播延时信息, 由于时序约束文件中设置了时钟本身的传播延时为实际时钟树的延时, 因此其传播延时值为 2.038ns。

(4) Setup

Setup 为终止点时序单元自身的建立时间信息, 其建立时间值为 0.131ns。

(5) Phase Shift

Phase Shift 为终止点时序单元用于时序计算的有效时钟触发沿与时钟源有效沿的间隔值。因为时序路径传播间隔约束为一个时钟周期，因此终止点单元的时钟有效沿间隔为时钟源有效沿后一个时钟周期的时间 20ns。

(6) CPPR Adjustment

CPPR Adjustment 为该时序路径中的 CPPR 值，这里为 0.052。

(7) Uncertainty

Uncertainty 为时序约束信息文件中定义的时钟不确定性值，与该时序路径相关的时钟不确定性值为 0.25ns。

(8) Clock Rise Edge

Clock Rise Edge 为始发点时序单元用于时序计算的有效时钟触发沿与时钟源有效沿的间隔值。由于始发点时序单元用于时序计算的有效时钟触发沿为时钟源有效沿，因此其值为 0。

(9) Clock Network Latency

Clock Network Latency 为发射时钟的自身传播延时信息。由于时序约束文件中设置了时钟本身的传播延时为实际时钟树的延时，其传播延时值为 2.263ns，同时通过附加信息 (Prop) 进行标记，如果没有进行该设置则默认时钟为理想时钟，那么附加信息为 (Ideal)。

(10) Beginpoint Arrival Time

Beginpoint Arrival Time 为 Clock Rise Edge 值与 Clock Network Latency 值之和 (Clock Rise Edge + Clock Network Latency)，这里值为 $0+2.263=2.263\text{ns}$ 。

(11) Required Time

Required Time 为满足时序要求、时序路径的延时不得超过的最大延时极限值，该极限值是基于建立时间计算公式并根据以上所列的时序参数信息计算出来的。

该条触发器到触发器时序路径的建立时间计算公式为：

$$\begin{aligned} \text{Required Time} = & \text{Other End Arrival Time} - \text{Setup} + \text{Phase Shift} \\ & + \text{CPPR Adjustment} - \text{Uncertainty} = 21.708\text{ns} \end{aligned}$$

(12) Arrival Time

Arrival Time 是该时序路径所用的实际延时值，该实际延时值具体信息在详细时序路径报告部分中说明。

(13) Slack Time

Slack Time 为 Required Time 值减去 Arrival Time 值的差值 (Required Time-Arrival Time)，为时序违反信息值。这里时序违反值为 0.099ns，正值说明没有建立时间时序违反。

在详细时序路径报告中，逐个顺序地列出了在从始发点时序单元到终止点时序单元的整条时序路径中所有的逻辑单元信息，同时对时序路径上的所有逻辑单元的时序信息进行了详细的说明，其中主要包含如下时序分析信息：

(14) Instance

Instance 用于说明该时序路径中的组成逻辑单元在设计门级网表中的实例化名称，比如

始发点单元实例化名称为: c/fpu_add/fpu_add_ctl/i_a2stg_fracadd_frac2/q_reg[0]。

(15) Arc

Arc 为时序路径中逻辑单元的时序弧信息, 如始发点单元 (/fpu_add/fpu_add_ctl/i_a2stg_fracadd_frac2/q_reg[0]) 的时序弧信息为 (CK ^-> Q v), 表示该单元的时序弧为从端口 CK 的 rising_edge 到端口 Q 的 falling_edge 时序弧, 这里 ^ 代表 rising_edge, v 代表 falling_edge。

(16) Cell

Cell 为时序路径中组成逻辑单元名称, 比如根据对应单元库的描述信息, DFFX4 说明是一个名为 DFF 同时其驱动能力为 X4 的时序单元; BUFX16 说明是一个名为 BUF 同时其驱动能力为 X16 的缓冲器单元; INVX1 说明是一个名为 INV 同时其驱动力为 X1 的反相器单元。

(17) Delay

Delay 为该条时序路径中单元的延时信息, 通过累加时序路径上所有单元的延时信息的方法, 可以计算出实际的 Arrival Time 值。

(18) Arrival Time

Arrival Time 前面已经介绍过, 在时序路径中每个单元都有对应的从始发点单元到达该单元的实际时序段延时值, 用于通过累加时序路径各个单元延时值方法来计算最终的时序延时值, 因此除了从始发点单元到终止点单元时序路径的 Arrival Time 为最终的完整时序延时值以外, 其他单元的 Arrival Time 为对应单元的局部累加时序延时值。

(19) Required Time

Required Time 前面已经介绍过, 在时序路径中每个单元都有对应的从始发点单元到达该单元的理想延时值, 理想延时为可以满足时序要求的最大延时极限值。由于时序延时值是通过累加时序路径各个单元延时值方法来计算的, 因此除了从始发点单元到终止点单元时序路径的 Required Time 为最终的完整最大延时极限值以外, 其他单元的 Arrival Time 为到达该单元的局部累加最大延时极限值。

4. 触发器到输出端时序路径分析

触发器到输入端时序路径表示从始发点为时序单元时钟输入端到终止点为组合逻辑单元的输出端口之间的时序路径, 因此其分析报告的主要特征是记录时序单元到输出端口之间的时序路径信息。

同样, 时序分析之前需要先定义时序约束信息, 以下是比较基本的与触发器到输出端时序路径分析相关的时序约束:

```
create_clock [get_pins c/gclk] -name gclk -period 20 -waveform {0 10}
set_clock_latency 0.1 [get_clocks gclk]
set_clock_uncertainty 0.25 [get_clocks gclk]
set_clock_transition -fall 0.15 [get_clocks gclk]
set_clock_transition -rise 0.15 [get_clocks gclk]
set_propagated_clock [all_clocks]
```

以上时序约束信息设置的大部分信息在前面已经介绍过, 这里不重复介绍。

设置与触发器到输出端时序路径的输出端口延时。这里通过其中一条输出端口的设置为例，设置输出端口 fp_cpx_req_cq_Top[7] 的延时为 0.15ns。

```
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[7]}]
```

OpenSparcT1 里浮点计算单元部件设计中所有的输出端口都需要有相对应的输入延时时序约束设置，如图 9-29 所示。

```
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[7]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[6]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[5]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[4]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[3]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[2]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[1]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[0]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[20]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[19]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[18]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[17]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[16]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[15]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[14]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[13]}]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_data_ca_Top[12]}]
```

图 9-29 输入延时时序约束设置

基于以上的时序约束信息，如下所示为 ETS 软件生成的关于这一条触发器到输出端时序路径的建立时间时序分析报告。

```
Endpoint: fp_cpx_req_cq_Top[7](v) checked with leading edge of 'gclk'
Beginpoint: c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7]/Q (v) triggered by
leading edge of 'gclk'
Other End Arrival Time          0.000
- External Delay                 0.150
+ Phase Shift                   20.000
+ CPPR Adjustment               0.000
- Uncertainty                   0.250
= Required Time                 19.600
- Arrival Time                  4.279
= Slack Time                    15.321

Clock Rise Edge                 0.000
+ Clock Network Latency (Prop)  2.119
= Beginpoint Arrival Time       2.119
```

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|--|-------------|---------|-------|--------------|---------------|
| c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7] | CK ^ | | | 2.119 | 17.440 |
| c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7] | CK ^ -> Q v | DFFHQX1 | 0.316 | 2.435 | 17.756 |
| c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/FE_OF4224_fp_cpx_req_cq_7_ | A v -> Y v | BUF8X8 | 0.200 | 2.635 | 17.956 |

```

c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/FE_OFC4223_fp_cpx_req_cq_7_
      A v -> Y v   BUF8X      0.228      2.863      18.184
B38      I v -> PAD v   PO8N      1.416      4.279      19.600
fp_cpx_req_cq_Top[7] v      0.000      4.279      19.600

```

总结报告中的部分时序分析信息在前面已经介绍过, 因此这里只对与触发器到输出端时序路径分析相关的分析信息进行说明。

(1) Endpoint

Endpoint 为输出端口的相关信息, 它包括终止点端口信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 `fp_cpx_req_cq_Top[7]` 的端口, 其信号输入端口的时序弧类型信息为 `falling_edge(v)`, 相关时钟为 `gclk`。

(2) Beginpoint

Beginpoint 为触发器到输出端时序路径中与发射时钟 (launch clock) 相关的时序单元的相关信息, 它包括始发点单元信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 `c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7]` 的单元, 其信号输出端口 Q 端的时序弧类型信息为 `falling_edge(v)`, 相关时钟为 `gclk`。

(3) External Delay

External Delay 为时序约束信息文件中定义的输出端口 `fp_cpx_req_cq_Top[7]` 的延时, 其值为 `0.15ns`。

(4) CPPR Adjustment

CPPR Adjustment 为该时序路径中的 CPPR 值, 由于触发器到输出端的时序路径中不存在公共的时钟树路径, 因此这里为 `0ns`。

(5) Required Time

该条触发器到输出端时序路径的建立时间计算公式为:

$$\begin{aligned} \text{Required Time} = & \text{Other End Arrival Time} - \text{External Delay} + \text{Phase Shift} \\ & + \text{CPPR Adjustment} - \text{Uncertainty} = 19.6\text{ns} \end{aligned}$$

(6) Slack Time

Slack Time 为 Required Time 值减去 Arrival Time 值的差值 (Required Time-Arrival Time), 为时序违反信息值。这里时序违反值为 `15.321ns`, 正值说明没有建立时间时序违反。

详细时序路径报告在前面已经介绍过, 除了时序路径的单元和延时不同以外, 其计算方法基本保持一致, 这里不重复介绍。

5. 输入端到触发器时序路径分析

输入端到触发器时序路径表示从始发点为组合逻辑单元的输入端口到终止点为时序单元时钟输入端之间的时序路径, 因此其分析报告主要特征是记录输入端口到时序单元之间的时序路径信息。

时序分析之前先需要定义时序约束信息, 以下是比较基本的与输入端到触发器时序路径

分析相关时序约束：

```
create_clock [get_pins c/gclk] -name gclk -period 20 -waveform {0 10}
set_clock_latency 0.1 [get_clocks gclk]
set_clock_uncertainty 0.25 [get_clocks gclk]
set_clock_transition -fall 0.15 [get_clocks gclk]
set_clock_transition -rise 0.15 [get_clocks gclk]
set_propagated_clock [all_clocks]
```

以上时序约束信息设置的大部分信息在前面已经介绍过，这里不重复介绍。

设置与输入端到触发器时序路径的输入端口延时，这里通过其中一条输入端口的设置为例，设置输入端口 `pcx_fpio_data_rdy_px2_Top` 的延时为 0.15ns。

```
set_input_delay -clock gclk 0.15 [get_ports pcx_fpio_data_rdy_px2_Top]
```

OpenSparcT1 里浮点计算单元部件设计中所有的输入端口都需要有相对应的输入延时时序约束设置，如图 9-30 所示。

```
set_input_delay -clock gclk 0.15 [get_ports pcx_fpio_data_rdy_px2_Top]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[20]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[19]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[18]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[17]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[16]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[15]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[14]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[13]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[12]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[11]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[10]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[9]}]
set_input_delay -clock gclk 0.15 [get_ports {pcx_fpio_data_px2_Top[8]}]
```

图 9-30 输入延时时序约束设置

基于以上的时序约束信息，如下所示为 ETS 软件生成的关于这一条触发器到输出端时序路径的建立时间时序分析报告。

```
Endpoint: c/fpu_in/fpu_in_ctl/i_fp_data_rdy/q_reg[0]/D (^) checked with leading
edge of 'gclk'
Beginpoint: pcx_fpio_data_rdy_px2_Top (^) triggered by leading edge of 'gclk'
Other End Arrival Time          2.188
- Setup                        0.170
+ Phase Shift                  20.000
+ CPPR Adjustment              0.000
- Uncertainty                  0.250
= Required Time                21.768
- Arrival Time                 2.936
= Slack Time                   18.832

  Clock Rise Edge              0.000
+ Input Delay                  0.150
= Beginpoint Arrival Time      0.150
-----
```

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|--|--------------|---------|-------|--------------|---------------|
| pcx_fpio_data_rdy_px2_Top ^ | | | | 0.150 | 18.982 |
| A1 | PAD ^ -> C ^ | PINN | 0.852 | 1.002 | 19.834 |
| FE_OFC845_pcx_fpio_data_rdy_px2 | | | | | |
| | A ^ -> Y ^ | ·BUFX8 | 0.231 | 1.233 | 20.064 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/U2 | | | | | |
| | AN ^ -> Y ^ | NOR3BX4 | 0.201 | 1.433 | 20.265 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/FE_PHC802_n1 | | | | | |
| | A ^ -> Y ^ | ·BUFXL | 0.141 | 1.574 | 20.405 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/FE_PHC5_n1 | | | | | |
| | A ^ -> Y ^ | DLY4X1 | 1.116 | 2.690 | 21.522 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/FE_OFC335_n1 | | | | | |
| | A ^ -> Y ^ | ·BUFX8 | 0.236 | 2.926 | 21.758 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/q_reg[0] | | | | | |
| | D ^ | DFFHQX2 | 0.010 | 2.936 | 21.768 |

总结报告中的部分时序分析信息在前面已经介绍过,因此这里只对与输入端到触发器时序路径分析相关的分析信息进行说明。

(1) Endpoint

Endpoint 为输入端到触发器时序路径中与捕获时钟相关的时序单元的相关信息,它包括终止点单元信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_in/fpu_in_ctl/i_fp_data_rdy/q_reg[0] 的单元,其信号输入端口 D 端的时序弧类型信息为 rising_edge (^),相关时钟为 gclk。

(2) Beginpoint

Beginpoint 为输入端口的相关信息,它包括始发点端口信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 pcx_fpio_data_rdy_px2_Top 的单元,其信号输入端口的时序弧类型信息为 rising_edge (^),相关时钟为 gclk。

(3) Input Delay

Input Delay 为时序约束信息文件中定义的输入端口 pcx_fpio_data_rdy_px2_Top 的延时,其值为 0.15ns。

(4) CPPR Adjustment

CPPR Adjustment 为该时序路径中的 CPPR 值,由于输入端到触发器的时序路径中不存在公共的时钟树路径,因此这里为 0ns。

(5) Beginpoint Arrival Time

Beginpoint Arrival Time 为 Clock Rise Edge 值与 Input Delay 值之和 (Clock Rise Edge + Input Delay),这里值为 0+0.15=0.15ns。

(6) Required Time

该条触发器到输出端时序路径的建立时间计算公式为:

$$\begin{aligned} \text{Required Time} &= \text{Other End Arrival Time} - \text{Setup} + \text{Phase Shift} \\ &\quad + \text{CPPR Adjustment} - \text{Uncertainty} = 21.768\text{ns} \end{aligned}$$

(7) Slack Time

Slack Time 为 Required Time 值减去 Arrival Time 值的差值 (Required Time-Arrival Time), 为时序违反信息值。这里时序违反值为 18.832ns, 正值说明没有建立时间时序违反。

详细时序路径报告在前面已经介绍过, 除了时序路径的单元和延时不同以外, 其计算方法基本保持一致, 这里不重复介绍。

6. 输入端到输出端时序路径分析

输入端到输出端时序路径表示从始发点为组合逻辑单元输入端口到终止点为组合逻辑单元输入端之间的时序路径。因此其分析报告主要特征是记录输入端口到输出端口之间的时序路径信息。

时序分析之前需要先定义时序约束信息, 以下是比较基本的与输入端到输出端时序路径分析相关时序约束:

(1) 设置时钟周期为 20ns 的 vclk 虚拟时钟, 相位占空比为 50%。

```
create_clock -name vclk -period 20 -waveform {0 10}
```

(2) 设置虚拟时钟信号的延时为 0.1ns。

```
set_clock_latency 0.1 [get_clocks vclk]
```

(3) 设置 vclk 虚拟时钟不确定性的数值为 0.25ns, 该设置主要是为了考虑时钟本身所带来的延时开销。

```
set_clock_uncertainty 0.25 [get_clocks vclk]
```

(4) 设置虚拟时钟的转换时间, 这里设置上拉和下拉转换时间都为 0.15ns。

```
set_clock_transition -fall 0.15 [get_clocks vclk]
set_clock_transition -rise 0.15 [get_clocks vclk]
```

(5) 设置输入端口延时。由于输入端到输出端时序路径存在输入端口, 因此需要设置输入端口延时。这里通过其中一条输入端到输出端时序路径的设置为例, 设置输入端口 c/fpu_add/fpu_add_frac_dp/U4928/B 的延时为 0.15ns。

```
set_input_delay -clock vclk 0.15 [get_pins c/fpu_add/fpu_add_frac_dp/U4928/B]
```

(6) 设置输出端口延时。由于输入端到输出端时序路径存在输出端口, 因此需要设置输出端口延时。这里通过其中一条输入端到输出端时序路径的设置为例, 设置输出端口 c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398/Y 的延时为 0.15ns。

```
set_output_delay -clock vclk 0.15 [get_pins c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398/Y]
```

基于以上的时序约束信息, 如下所示为 ETS 软件生成的这一条输入端到输出端时序路径的建立时间时序分析报告。

```

Endpoint: c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398/Y (v)
checked with leading edge of 'vclk'
Beginpoint: c/fpu_add/fpu_add_frac_dp/U4928/B (v)
triggered by leading edge of 'vclk'
Other End Arrival Time          0.000
+ Network Insertion Delay       0.100
- External Delay                0.150
+ Phase Shift                   20.000
+ CPPR Adjustment              0.000
- Uncertainty                   0.250
= Required Time                 19.700
- Arrival Time                  9.027
= Slack Time                    10.673

    Clock Rise Edge              0.000
    + Input Delay                0.150
    + Network Insertion Delay    0.100
    = Beginpoint Arrival Time    0.250

```

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|--|-------------|-----------|-------|--------------|---------------|
| c/fpu_add/fpu_add_frac_dp/U4928 | B v | | | 0.250 | 10.923 |
| c/fpu_add/fpu_add_frac_dp/U4928 | B v -> Y ^ | MXI2X2 | 0.148 | 0.398 | 11.072 |
| c/fpu_add/fpu_add_frac_dp/U4927 | B0 ^ -> Y v | OAI21X1 | 0.105 | 0.503 | 11.177 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/FE_OFC5571_a2stg_fracadd_in2_0_ | A v -> Y ^ | INVX1 | 0.121 | 0.624 | 11.298 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/FE_OFC5572_a2stg_fracadd_in2_0_ | A ^ -> Y v | INVX3 | 0.091 | 0.715 | 11.388 |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| (时序路径中间部分信息省略) | | | | | |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U402 | B0 ^ -> Y v | OAI2BB1X2 | 0.139 | 8.717 | 19.391 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U399 | A1 v -> Y ^ | OAI21X2 | 0.172 | 8.890 | 19.563 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398 | B0 ^ -> Y v | OAI2BB1X2 | 0.137 | 9.027 | 19.700 |

总结报告中的部分时序分析信息在前面已经介绍过，因此这里对与输入端到输出端时序路径分析相关的分析信息进行说明。

(1) Endpoint

Endpoint 为输出端口的相关信息，它包括终止点端口信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/

U398 的 Y 端口, 其信号输入端口的时序弧类型信息为 falling_edge(v), 相关时钟为 vclk。

(2) Beginpoint

Beginpoint 为输入端口的相关信息, 它包括始发点端口信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_frac_dp/U4928 的 B 端口, 其信号输入端口的时序弧类型信息为 falling_edge(v), 相关时钟为 vclk。

(3) Input Delay

Input Delay 即为时序约束信息文件中定义的输入端口 c/fpu_add/fpu_add_frac_dp/U4928/B 的延时, 其值为 0.15ns。

(4) External Delay

External Delay 为时序约束信息文件中定义的输出端口 c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398/Y 的延时, 其值为 0.15ns。

(5) CPPR Adjustment

CPPR Adjustment 为该时序路径中的 CPPR 值, 由于输入端到输入端的时序路径中时钟为虚拟时钟, 因此这里为 0ns。

(6) Beginpoint Arrival Time

Beginpoint Arrival Time 为 Clock Rise Edge 值、Input Delay 值和 Network Insertion Delay 值之和 (Clock Rise Edge + Input Delay + Network Insertion Delay), 这里值为 $0+0.1+0.15=0.25\text{ns}$ 。

(7) Required Time

该条输入端到输出端时序路径的建立时间计算公式为:

$$\begin{aligned} \text{Required Time} = & \text{Other End Arrival Time} + \text{Network Insertion Delay} - \text{External Delay} \\ & + \text{Phase Shift} + \text{CPPR Adjustment} - \text{Uncertainty} = 19.7\text{ns} \end{aligned}$$

(8) Slack Time

Slack Time 为 Required Time 值减去 Arrival Time 值的差值 (Required Time-Arrival Time), 为时序违反信息值。这里时序违反值为 10.673ns, 正值说明没有建立时间时序违反。

详细时序路径报告在前面已经介绍过, 除了时序路径的单元和延时不同以外, 其计算方法基本保持一致, 这里不重复介绍。

9.3.2 保持时间分析

保持时间分析是分析为保证时序单元对数据读取正确, 数据在时钟有效沿到来之后, 仍需要保持稳定的时间是否满足要求。与建立时间分析一样, 保持时间分析中也需要分别对 4 种类型的时序路径进行分析。

1. 保持时间分析模式设置

ETS 中进行保持时间分析之前需要定义静态时序分析模式, 在菜单栏里单击 Timing&SI → Specify Timing Analysis Mode, 如图 9-31 所示。

静态时序分析模式设置界面有两个分类设置栏: Basic 设置栏和 Advanced 设置栏, 这里主要是对 Basic 设置栏进行设置, Advanced 设置栏保持默认值。由于 Basic 栏是默认显示的, 其设置界面如图 9-32 所示。

在 Timing Check 设置框中, 选择 Hold 选项来定义进行保持时间分析; 在 Timing Mode 设置框中选择 Best and Worst Case 选项来定义静态时序分析方法基于 WC_BC 模式; 选中 CPPR 和 Clock Gating Check 复选项, 最后单击 OK 按钮完成静态时序分析模式的设置。

在静态时序分析模式设置中, 除了选择了 CPPR 设置, 还需要设置时序减免 (timing derate)。在菜单栏里单击 Timing&SI → Set Timing Derate, 如图 9-33 所示。

进入时序减免设置界面, 如图 9-34 所示。

在最好情况下 (Min), 由于发射时钟路径上传播延时的路径与捕获传播延时短的路径之间的延时差过于悲观, 因此这里设置 Late 选项中的值为 1.05, 使得时序分析结果更加合理。1.05 这个数值是根据流片工艺节点和工程经验等方面评估而来的, 因此不同的工艺和工程中时序减免值不一定相同。单击 OK 按钮完成时序减免的参数设置。

时序减免设置完后, 就可以通过 ETS 软件进行保持时间的时序计算, 并产生时序分析结果。在菜单栏里单击 Timing&SI → Timing Analysis, 如图 9-35 所示。

进入时序分析设置界面, 如图 9-36 所示。

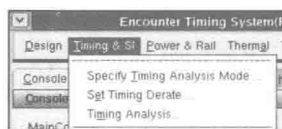


图 9-31 选择分析模式设置

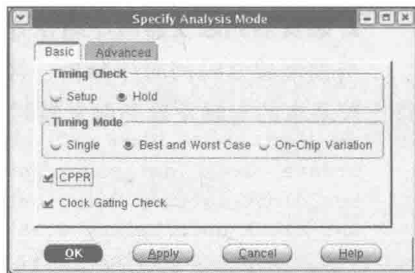


图 9-32 静态时序分析模式设置界面

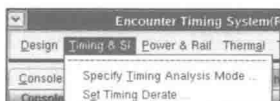


图 9-33 选择时序减免设置

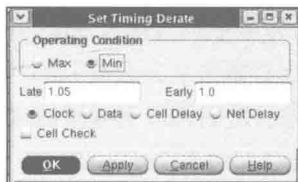


图 9-34 时序减免设置界面

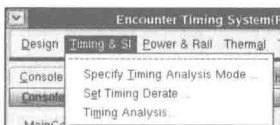


图 9-35 选择时序分析设置

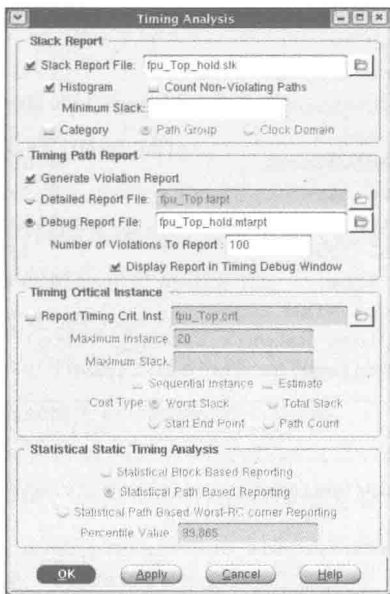


图 9-36 时序分析设置界面

在 Timing Path Report 设置栏中选中 Generate Violation Report 和 Debug Report File 这两个选项。在 Debug Report File 选项中设置正确的时序报告文件名称，并定义时序报告中所记录的时序路径数量，这里设置为 100。这里时序报告文件名称为 fpu_Top_hold.mtarpt，该报告文件提供给 ETS 工具内嵌的图形化时序分析器使用，这样可以帮助设计者进行基于图形界面的时序分析工作。最后单击 OK 按钮，ETS 软件就开始进行保持时间的时序计算，并产生时序分析结果。

2. 触发器到触发器时序路径分析

与前面建立时间时序分析一样，保持时间分析之前需要先定义相关时序约束信息，以下是比较基本的与触发器到触发器时序路径分析相关的时序约束：

```
create_clock [get_pins c/gclk] -name gclk -period 20 -waveform {0 10}
set_clock_latency 0.1 [get_clocks gclk]
set_clock_uncertainty 0.25 [get_clocks gclk]
set_clock_transition -fall 0.15 [get_clocks gclk]
set_clock_transition -rise 0.15 [get_clocks gclk]
set_propagated_clock [all_clocks]
```

以上时序约束信息设置的大部分信息在前面建立时间分析中已经介绍过，这里不重复介绍。

基于以上的时序约束信息，如下所示为 ETS 软件生成的关于这一条触发器到触发器时序路径的保持时间时序分析报告。

Endpoint: c/fpu_add/fpu_add_ctl/i_a4stg_nv2/q_reg[0]/D (v) checked with leading edge of 'gclk'
Beginpoint: c/fpu_add/fpu_add_ctl/i_a3stg_nv/q_reg[0]/Q (v) triggered by leading edge of 'gclk'

Other End Arrival Time 1.157
+ Hold -0.030
+ Phase Shift 0.000
- CPPR Adjustment 0.055
+ Uncertainty 0.250
= Required Time 1.322
Arrival Time 1.331
Slack Time 0.009
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 1.101
= Beginpoint Arrival Time 1.101

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|---|-------------|---------|-------|--------------|---------------|
| c/fpu_add/fpu_add_ctl/i_a3stg_nv/q_reg[0] | CK ^ | | | 1.101 | 1.092 |
| c/fpu_add/fpu_add_ctl/i_a3stg_nv/q_reg[0] | CK ^ -> Q v | DFFHQX2 | 0.143 | 1.244 | 1.235 |
| c/fpu_add/fpu_add_ctl/i_a4stg_nv2/U4 | B v -> Y ^ | MXI2X2 | 0.056 | 1.300 | 1.291 |

```

c/fpu_add/fpu_add_ctl/i_a4stg_nv2/U5
      B ^ -> Y v      NOR2X1      0.032      1.331      1.322
c/fpu_add/fpu_add_ctl/i_a4stg_nv2/q_reg[0]
      D v      FFHQX1      0.000      1.331      1.322

```

总结报告中的部分时序分析信息在前面建立时间分析中已经介绍过,因此这里只对与触发器到触发器时序路径分析相关的分析信息进行说明。

(1) Endpoint

Endpoint 为输出端口的相关信息,它包括终止点端口信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_ctl/i_a4stg_nv2/q_reg[0] 的单元,其信号输出端口 D 端的时序弧类型信息为 falling_edge(v),相关时钟为 gclk。

(2) Beginpoint

Beginpoint 为触发器到触发器时序路径中与发射时钟(Launch Clock)相关的时序单元的相关信息,它包括始发点单元信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_ctl/i_a3stg_nv/q_reg[0] 的单元,其信号输出端口 Q 端的时序弧类型信息为 falling_edge(v),相关时钟为 gclk。

(3) Hold

Hold 为终止点时序单元自身的保持时间信息,其建立时间值为 -0.030ns。

(4) Phase Shift

Phase Shift 为终止点时序单元用于时序计算的有效时钟触发沿与时钟源有效沿的间隔值。由于建立时间计算方法中终止点单元的时钟有效沿与时钟源有效沿同步,因此其值为 0。

(5) Beginpoint Arrival Time

Beginpoint Arrival Time 为 Clock Rise Edge 值与 Clock Network Latency 值之和 (Clock Rise Edge + Clock Network Latency),这里值为 0+1.101=1.101ns。

(6) Required Time

Required Time 为满足时序要求、时序路径的延时不得超过的最小延时极限值,该极限值是基于保持时间计算公式并根据以上所列的时序参数信息计算出来的。

该条触发器到触发器时序路径的保持时间计算公式为:

$$\begin{aligned} \text{Required Time} = & \text{Other End Arrival Time} + \text{Hold} + \text{Phase Shift} \\ & - \text{CPRR Adjustment} + \text{Uncertainty} = 1.322\text{ns} \end{aligned}$$

(7) Arrival Time

Arrival Time 是该时序路径所用的实际延时值,该实际延时值具体信息在详细时序路径报告部分中说明。

(8) Slack Time

Slack Time 为 Arrival Time 值减去 Required Time 值的差值 (Arrival Time-Required Time),

为时序违反信息值。这里时序违反值为 0.009ns，正值说明没有保持时间时序违反。

详细时序路径报告在前面已经介绍过，除了时序路径的单元和延时不同以外，其计算方法基本保持一致，这里不重复介绍。

3. 触发器到输出端时序路径分析

触发器到输出端时序路径保持时间分析之前需要先定义相关时序约束信息，以下是比较基本的与触发器到输出端时序路径分析相关的时序约束：

```
create_clock [get_pins c/gclk] -name gclk -period 20 -waveform {0 10}
set_clock_latency 0.1 [get_clocks gclk]
set_clock_uncertainty 0.25 [get_clocks gclk]
set_clock_transition -fall 0.15 [get_clocks gclk]
set_clock_transition -rise 0.15 [get_clocks gclk]
set_propagated_clock [all_clocks]
set_output_delay -clock gclk 0.15 [get_ports {fp_cpx_req_cq_Top[7]}]
```

以上时序约束信息设置的大部分信息在前面建立时间分析中已经介绍过，这里不重复介绍。

基于以上的时序约束信息，如下所示为 ETS 软件生成的这一条触发器到输出端时序路径的保持时间时序分析报告。

Endpoint: fp_cpx_req_cq_Top[7] (v) checked with leading edge of 'gclk'
Beginpoint: c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7]/Q (v) triggered by leading edge of 'gclk'

| | |
|--------------------------------|--------|
| Other End Arrival Time | 0.000 |
| - External Delay | 0.150 |
| + Phase Shift | -0.000 |
| - CPPR Adjustment | 0.000 |
| + Uncertainty | 0.250 |
| = Required Time | 0.100 |
| Arrival Time | 1.893 |
| Slack Time | 1.793 |
| Clock Rise Edge | 0.000 |
| + Clock Network Latency (Prop) | 1.010 |
| = Beginpoint Arrival Time | 1.010 |

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|---|--------------|---------|-------|--------------|---------------|
| c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7] | CK ^ | | | 1.010 | -0.782 |
| c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7] | CK ^ -> Q v | DFFHQX1 | 0.141 | 1.151 | -0.642 |
| c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/FE_OFC4224_fp_cpx_req_cq_7_ | A v -> Y v | BUF8 | 0.092 | 1.243 | -0.550 |
| c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/FE_OFC4223_fp_cpx_req_cq_7_ | A v -> Y v | BUF8 | 0.108 | 1.351 | -0.442 |
| B38 | I v -> PAD v | PO8N | 0.542 | 1.893 | 0.100 |
| fp_cpx_req_cq_Top[7] v | | | 0.000 | 1.893 | 0.100 |

总结报告中的部分时序分析信息在前面建立时间分析已经介绍过, 因此这里只对与触发器到输出端时序路径分析相关的分析信息进行说明。

(1) Endpoint

Endpoint 为输出端口的相关信息, 它包括终止点端口信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 `fp_cpx_req_cq_Top[7]` 的端口, 其信号输入端口的时序弧类型信息为 `falling_edge(v)`, 相关时钟为 `gclk`。

(2) Beginpoint

Beginpoint 为触发器到输出端时序路径中与发射时钟 (Launch Clock) 相关的时序单元的相关信息, 它包括始发点单元信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 `c/fpu_out/fpu_out_ctl/i_fp_cpx_req_cq/q_reg[7]` 的单元, 其信号输出端口 Q 端的时序弧类型信息为 `falling_edge(v)`, 相关时钟为 `gclk`。

(3) External Delay

External Delay 为时序约束信息文件中定义的输出端口 `fp_cpx_req_cq_Top[7]` 的延时, 其值为 `0.15ns`。

(4) CPPR Adjustment

CPPR Adjustment 为该时序路径中的 CPPR 值, 由于触发器到输出端的时序路径中不存在公共的时钟树路径, 因此这里为 `0ns`。

(5) Required Time

该条触发器到输出端时序路径的保持时间计算公式为:

$$\begin{aligned} \text{Required Time} = & \text{Other End Arrival Time} - \text{External Delay} + \text{Phase Shift} \\ & - \text{CPPR Adjustment} + \text{Uncertainty} = 0.1\text{ns} \end{aligned}$$

(6) Arrival Time

Arrival Time 是该时序路径所用的实际延时值, 该实际延时值具体信息在详细时序路径报告部分中说明。

(7) Slack Time

Slack Time 为 Arrival Time 值 减去 Required Time 值的 差值 (Arrival Time-Required Time), 为时序违反信息值。这里时序违反值为 `1.793ns`, 正值说明没有保持时间时序违反。

详细时序路径报告在前面已经介绍过, 除了时序路径的单元和延时不同以外, 其计算方法基本保持一致, 这里不重复介绍。

4. 输入端到触发器时序路径分析

输入端到触发器时序路径保持时间分析之前需要先定义相关时序约束信息, 以下是比较基本的与输入端到触发器时序路径分析相关的时序约束:

```
create_clock [get_pins c/gclk] -name gclk -period 20 -waveform {0 10}
set_clock_latency 0.1 [get_clocks gclk]
set_clock_uncertainty 0.25 [get_clocks gclk]
```

```
set_clock_transition -fall 0.15 [get_clocks gclk]
set_clock_transition -rise 0.15 [get_clocks gclk]
set_propagated_clock [all_clocks]
set_input_delay -clock gclk 0.15 [get_ports pcx_fpio_data_rdy_px2_Top]
```

以上时序约束信息设置的大部分信息在前面建立时间分析中已经介绍过，这里不重复介绍。

基于以上的时序约束信息，如下所示为 ETS 软件生成的这一条输入端到触发器时序路径的保持时间时序分析报告。

```
Endpoint: c/fpu_in/fpu_in_ctl/i_fp_data_rdy/q_reg[0]/D (v) checked with leading
edge of 'gclk'
Beginpoint: pcx_fpio_data_rdy_px2_Top (v) triggered by leading edge of 'gclk'
Other End Arrival Time          1.165
+ Hold                          -0.043
+ Phase Shift                   0.000
- CPPR Adjustment               0.000
+ Uncertainty                   0.250
= Required Time                 1.372
Arrival Time                    1.256
Slack Time                      -0.116

    Clock Rise Edge              0.000
    + Input Delay                0.150
    = Beginpoint Arrival Time    0.150
```

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|--|--------------|---------|-------|--------------|---------------|
| pcx_fpio_data_rdy_px2_Top v | | | | 0.150 | 0.266 |
| A1 | PAD v -> C v | PINN | 0.249 | 0.399 | 0.515 |
| FE_OFC845_pcx_fpio_data_rdy_px2 | | | | | |
| | A v -> Y v | BUF8X8 | 0.120 | 0.518 | 0.634 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/U2 | | | | | |
| | AN v -> Y v | NOR3BX4 | 0.104 | 0.623 | 0.739 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/FE_PHC802_n1 | | | | | |
| | A v -> Y v | BUF8X8 | 0.062 | 0.685 | 0.801 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/FE_PHC5_n1 | | | | | |
| | A v -> Y v | DLY4X1 | 0.447 | 1.133 | 1.248 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/FE_OFC335_n1 | | | | | |
| | A v -> Y v | BUF8X8 | 0.114 | 1.246 | 1.362 |
| c/fpu_in/fpu_in_ctl/i_fp_data_rdy/q_reg[0] | | | | | |
| | D v | DF8XQX2 | 0.010 | 1.256 | 1.372 |

(1) Endpoint

Endpoint 为输入端到触发器时序路径中与捕获时钟（Capture Clock）相关的时序单元的相关信息，它包括终止点单元信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_in/fpu_in_ctl/i_fp_data_rdy/q_reg[0] 的单元，其信号输入端口 D 端的时序弧类型信息为 falling_edge(v)，相关时钟为 gclk。

(2) Beginpoint

Beginpoint 为输入端口的相关信息, 它包括始发点端口信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 `pcx_fpio_data_rdy_px2_Top` 的单元, 其信号输入端口的时序弧类型信息为 `falling_edge(v)`, 相关时钟为 `gclk`。

(3) Hold

Hold 为终止点时序单元自身的保持时间信息, 其建立时间值为 -0.043ns 。

(4) Input Delay

Input Delay 为时序约束信息文件中定义的输入端口 `pcx_fpio_data_rdy_px2_Top` 的延时, 其值为 0.15ns 。

(5) CPPR Adjustment

CPPR Adjustment 为该时序路径中的 CPPR 值, 由于输入端到触发器的时序路径中不存在公共的时钟树路径, 因此这里为 0ns 。

(6) Beginpoint Arrival Time

Beginpoint Arrival Time 为 Clock Rise Edge 值与 Input Delay 值之和 (Clock Rise Edge + Input Delay), 这里值为 $0+0.15=0.15\text{ns}$ 。

(7) Required Time

该条输入端到触发器时序路径的保持时间计算公式为:

$$\begin{aligned} \text{Required Time} = & \text{Other End Arrival Time} + \text{Hold} + \text{Phase Shift} \\ & - \text{CPPR Adjustment} + \text{Uncertainty} = 1.372\text{ns} \end{aligned}$$

(8) Slack Time

Slack Time 为 Arrival Time 值减去 Required Time 值的差值 (Arrival Time-Required Time), 为时序违反信息值。这里时序违反值为 -0.116ns , 负值说明保持时间存在时序违反。

详细时序路径报告在前面已经介绍过, 除了时序路径的单元和延时不同以外, 其计算方法基本保持一致, 这里不重复介绍。

5. 输入端到输出端时序路径分析

输入端到输出端时序路径保持时间分析之前需要先定义相关时序约束信息, 以下是比较基本的与输入端到输出端时序路径分析相关的时序约束:

```
create_clock -name vclk -period 20 -waveform {0 10}
set_clock_latency 0.1 [get_clocks vclk]
set_clock_uncertainty 0.25 [get_clocks vclk]
set_clock_transition -fall 0.15 [get_clocks vclk]
set_clock_transition -rise 0.15 [get_clocks vclk]
set_input_delay -clock vclk 0.15 [get_pins c/fpu_add/fpu_add_frac_dp/U4928/B]
set_output_delay -clock vclk 0.15 [get_pins c/fpu_add/fpu_add_frac_dp/add_1_
root_add_677_2/U398/Y]
```

以上时序约束信息设置的大部分信息在前面建立时间分析中已经介绍过, 这里不重复介绍。

基于以上的时序约束信息, 如下所示为 ETS 软件生成的这一条输入端到输出端时序路径的保持时间时序分析报告。

Endpoint: c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398/Y (^) checked with leading edge of 'vclk'

Beginpoint: c/fpu_add/fpu_add_frac_dp/U4928/B (^) triggered by leading edge of 'vclk'

| | |
|---------------------------|-------|
| Other End Arrival Time | 0.000 |
| + Network Insertion Delay | 0.100 |
| - External Delay | 0.150 |
| + Phase Shift | 0.000 |
| - CPPR Adjustment | 0.000 |
| + Uncertainty | 0.250 |
| = Required Time | 0.200 |
| Arrival Time | 3.394 |
| Slack Time | 3.194 |

| | |
|---------------------------|-------|
| Clock Rise Edge | 0.000 |
| + Input Delay | 0.150 |
| + Network Insertion Delay | 0.100 |
| = Beginpoint Arrival Time | 0.250 |

| Instance | Arc | Cell | Delay | Arrival Time | Required Time |
|--|-------------|-----------|-------|--------------|---------------|
| c/fpu_add/fpu_add_frac_dp/U4928 | B ^ | | | 0.250 | -2.944 |
| c/fpu_add/fpu_add_frac_dp/U4928 | B ^ -> Y v | MXI2X2 | 0.054 | 0.304 | -2.890 |
| c/fpu_add/fpu_add_frac_dp/U4927 | B0 v -> Y ^ | OAI21X1 | 0.038 | 0.342 | -2.852 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/FE_OFC5571_a2stg_fracadd_in2_0_ | A ^ -> Y v | INVX1 | 0.037 | 0.380 | -2.815 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/FE_OFC5572_a2stg_fracadd_in2_0_ | A v -> Y ^ | INVX3 | 0.055 | 0.434 | -2.760 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U505 | A1 ^ -> Y v | OAI21X4 | 0.038 | 0.472 | -2.722 |
| (时序路径中间部分信息省略) | | | | | |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U402 | B0 v -> Y ^ | OAI2BB1X2 | 0.074 | 3.281 | 0.087 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U399 | A1 ^ -> Y v | OAI21X2 | 0.040 | 3.321 | 0.127 |
| c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398 | B0 v -> Y ^ | OAI2BB1X2 | 0.073 | 3.394 | 0.200 |

总结报告中的部分时序分析信息在前面已经介绍，因此这里只对与输入端到输出端时序路径分析相关的分析信息进行说明。

(1) Endpoint

Endpoint 为输出端口的相关信息，它包括终止点端口信息、信号输入端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398 的 Y 端口，其信号输入端口的时序弧类型信息为 rising_edge (^)，相关时钟为 vclk。

(2) Beginpoint

Beginpoint 为输入端口的相关信息, 它包括始发点端口信息、信号输出端口时序弧信息和相关时钟信息。这里终止点信息为名为 c/fpu_add/fpu_add_frac_dp/U4928 的 B 端口, 其信号输入端口的时序弧类型信息 rising_edge (^), 相关时钟为 vclk。

(3) Input Delay

Input Delay 为时序约束信息文件中定义的输入端口 c/fpu_add/fpu_add_frac_dp/U4928/B 的延时, 其值为 0.15ns。

(4) External Delay

External Delay 为时序约束信息文件中定义的输出端口 c/fpu_add/fpu_add_frac_dp/add_1_root_add_677_2/U398/Y 的延时, 其值为 0.15ns。

(5) CPPR Adjustment

CPPR Adjustment 为该时序路径中的 CPPR 值, 由于输入端到输入端的时序路径中时钟为虚拟时钟, 因此这里为 0ns。

(6) Beginpoint Arrival Time

Beginpoint Arrival Time 为 Clock Rise Edge 值、Input Delay 值和 Network Insertion Delay 值之和 (Clock Rise Edge + Input Delay + Network Insertion Delay), 这里值为 $0+0.1+0.15=0.25\text{ns}$ 。

(7) Required Time

该条输入端到输出端时序路径的保持时间计算公式为:

$$\begin{aligned} \text{Required Time} = & \text{Other End Arrival Time} + \text{Network Insertion Delay} - \text{External Delay} \\ & + \text{Phase Shift} - \text{CPPR Adjustment} + \text{Uncertainty} = 0.2\text{ns} \end{aligned}$$

(8) Slack Time

Slack Time 为 Arrival Time 值减去 Required Time 值的差值 (Arrival Time-Required Time), 为时序违反信息值。这里时序违反值为 3.194ns, 正值说明没有建立时间时序违反。

详细时序路径报告在前面已经介绍过, 除了时序路径的单元和延时不同以外, 其计算方法基本保持一致, 这里不重复介绍。

9.3.3 时序设计规则分析

时序路径除了满足建立时间和保持时间的时序要求以外, 也需要满足时序设计规则的要求, 因此还要对进行时序设计规则检查。前面已经介绍过时序设计规则主要包括 3 个方面: 最大转换时间、最大电容负载、最大扇出。

基于前面建立时间工作环境, 在工作界面选择栏中单击 Analysis 选项进入图形化时序分析界面, 单击 Analysis → DRV Analysis, 如图 9-37 所示。

进入 DRV 设置界面后, 在 Option 设置框中选择需要生

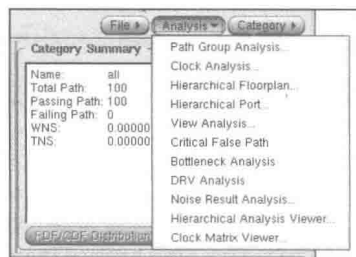


图 9-37 选择 DRV 分析设置

成的时序设计规则报告并填入正确的报告名称。这里选择产生 3 种设计规则报告，同时最大转换时间报告命名为 tranViolation.rpt，时间最大电容负载报告命名为 capViolation.rpt，最大扇出报告命名为 fanoutViolation.rpt，如图 9-38 所示。

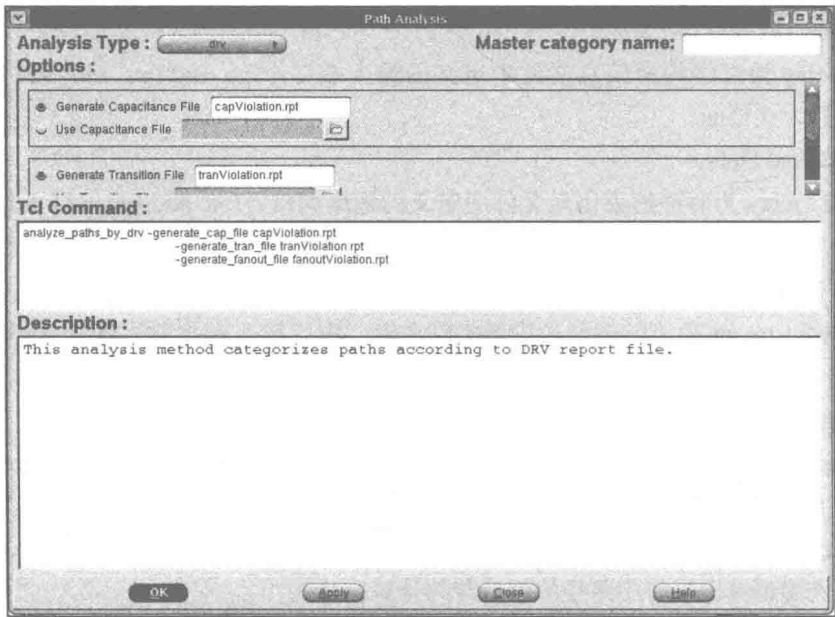


图 9-38 DRV 设置界面

单击 OK 按钮生成时序设计规则文件。
设计中需要设置时序设计规则约束来保证逻辑单元能够正常工作，因此时序设计规则分析之前需要先定义时序设计规则约束信息，以下是比较基本的时序设计规则约束：

- 1) 设置最大转换时间为 0.25ns。

```
set_max_transition 0.25 [current_design]
```

基于以上的最大转换时间约束信息，如下所示为 ETS 软件生成的部分关于最大转换时间的分析报告。

| InstPin | MaxTranTime | TranTime | TranSlack | CellPort |
|--|---------------|---------------|-----------------|------------|
| c/i_fpu_inq_sram/U7687/A1 | 0.250r/0.250f | 0.325r/0.270f | -0.075r/-0.020f | AOI22X1/A1 |
| c/i_fpu_inq_sram/U9144/A1 | 0.250r/0.250f | 0.325r/0.270f | -0.075r/-0.020f | AOI22X1/A1 |
| c/i_fpu_inq_sram/inq_ary_reg[4][137]/Q | 0.250r/0.250f | 0.325r/0.270f | -0.075r/-0.020f | TLATX1/Q |
| c/i_fpu_inq_sram/U7655/A1 | 0.250r/0.250f | 0.325r/0.270f | -0.075r/-0.020f | AOI22X1/A1 |

在以上所示的最大转换时间违反报告主要包含如下时序分析信息:

(1) InstPin

InstPin 为设计网表中存在转换时间违反的实例化单元端口信息。比如 c/i_fpu_inq_sram/U7687/A1, 这里实例化单元为 c/i_fpu_inq_sram/U7687, 其端口为 A1。

(2) MaxTranTime

MaxTranTime 为设计中最大转换时间的设置信息, 这里设置的是信号上拉和信号下拉转换时间都为 0.25ns。比如 0.250r/0.250f, 其中 r (Rising) 代表信号上拉转换时间, f (Falling) 代表信号下拉转换时间。

(3) TranTime

TranTime 为设计网表中存在转换时间违反的单元端口的实际转换时间, 比如 0.325r/0.270f, 其中包括信号上拉转换时间信息和信号下拉转换时间信息。

(4) TranSlack

TranSlack 为设计网表中存在转换时间违反的单元端口的转换时间违反值, 比如 -0.075r/-0.020f, 同样包括信号上拉转换时间违反值信息和信号下拉转换时间违反值信息。

(5) CellPort

CellPort 为对应的用于组成设计网表中实例化单元的实际逻辑门单元端口。比如 AOI22X1/A1, 这里是与或非门 AOI22X1 逻辑门单元, 其端口为 A1。

2) 设置最大电容负载为 0.5pF。

```
set_max_capacitance 0.5 [current_design]
```

基于以上的最大电容负载约束信息, 如下所示为 ETS 软件生成的部分关于最大电容负载分析报告。

| # InstPin | MaxCap | Cap | CapSlack | CellPort |
|-------------------------------|--------|-------|----------|-----------|
| c/i_fpu_inq_sram/ren_d1_reg/Q | 0.500 | 5.354 | -4.854 | EDFFX2/Q |
| c/i_fpu_inq_sram/U5905/Y | 0.500 | 2.981 | -2.481 | NAND2X2/Y |
| c/i_fpu_inq_sram/U5903/Y | 0.500 | 2.981 | -2.481 | NAND2X2/Y |
| c/i_fpu_inq_sram/U5901/Y | 0.500 | 2.789 | -2.289 | CLKINX3/Y |

总结报告中的 InstPin 和 CellPort 在前面已经介绍过, 因此这里只对与最大电容负载分析相关的其他信息进行说明。

(1) MaxCap

MaxCap 为设计中最大电容负载的设置信息, 这里最大电容负载值设置为 0.5pF。

(2) Cap

Cap 为设计中存在最大电容负载违反的实例化单元端口的实际电容负载值，比如 5.354pF。

(3) CapSlack

CapSlack 为设计网表中存在最大电容负载违反的实例化单元端口的电容负载违反值，比如 -4.854pF。

3) 设置最大扇出为 48。

```
set_max_fanout 48 [current_design]
```

基于以上的最大电容负载约束信息，如下所示为 ETS 软件生成的部分关于最大电容负载分析报告。

| # | InstPin | MaxFanLoad | FanLoad | FanLoadSlk | CellPort |
|--------------------------|---------|------------|---------|------------|----------|
| c/i_fpu_inq_sram/U5940/Y | | 48.000 | 51.000 | -3.000 | INVX1/Y |
| c/i_fpu_inq_sram/U5936/Y | | 48.000 | 50.000 | -2.000 | INVX1/Y |
| c/i_fpu_inq_sram/U5935/Y | | 48.000 | 50.000 | -2.000 | INVX1/Y |
| c/i_fpu_inq_sram/U5946/Y | | 48.000 | 50.000 | -2.000 | INVX1/Y |

在以上所示的单元输出端口最大扇出值违反报告主要包含如下时序分析信息，总结报告中的 InstPin 和 CellPort 在前面已经介绍过，因此这里只对与最大电容负载分析相关的其他信息进行说明。

(1) MaxFanLoad

MaxFanLoad 为设计中单元输出端口最大扇出值的设置信息，这里最大扇出值设置为 48。

(2) FanLoad

FanLoad 为设计中存在最大扇出值违反的实例化单元端口的实际扇出值，比如 51。

(3) FanLoadSlk

FanLoadSlk 为设计网表中存在最大扇出值违反的实例化单元端口的扇出违反值，比如 -3。

9.3.4 时序违反修复

静态时序分析不仅能够分析出不满足时序约束的时序路径，也可以根据时序分析报告进行时序优化，来修复存在时序违反的时序路径。这里介绍基于 ETS 工具的时序优化来进行时序违反修复的基本过程。

ETS 中进行时序优化需要进入时序违反修复设置界面，首先在菜单栏里单击 Timing&SI → Interactive ECO，如图 9-39 所示。

时序违反修复设置界面如图 9-40 所示。

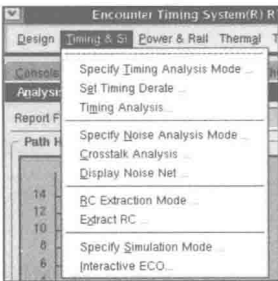


图 9-39 选择 ECO 设置

在时序违反修复设置界面中有 4 个设置选项: Add Repeater、Change Cell、Del Repeater 和 Save ECO History。可以看出, ETS 时序工具提供 3 种时序优化方法来进行时序违反修复, 它们分别对应时序违反修复设置界面 4 个设置选项的前 3 个: 插入缓冲器单元 (Add Repeater)、改变单元大小 (Change Cell) 和删除缓冲器单元 (Del Repeater)。第 4 个设置选项为保存 ECO 历史 (Save ECO History), 该设置用于保存时序违反修复过程中所执行的时序优化命令信息。

1. 插入缓冲器单元

插入缓冲器单元的方法是时序优化中最典型的方法之一, 时序违反修复中插入缓冲器单元设置界面如图 9-41 所示。

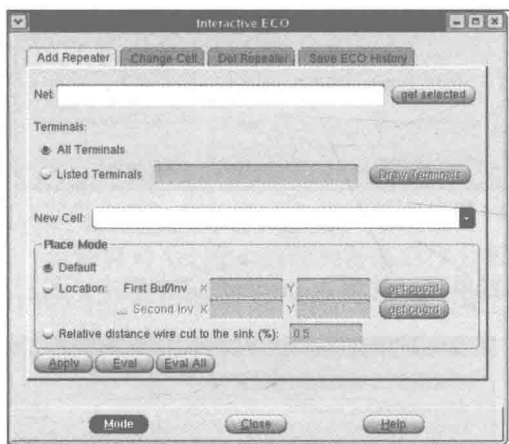


图 9-40 时序违反修复设置界面

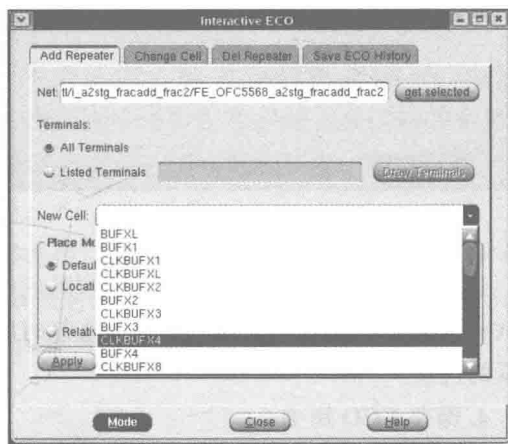


图 9-41 插入缓冲器单元设置界面

通过在时序分析报告中确定需要插入缓冲器单元的连线, 并填入 Net 文本框中, 同时在 New Cell 设置框中选择合适的缓冲器单元。

最后单击 Apply 按钮即可完成插入缓冲器单元修复时序违反的过程。

2. 改变单元大小

改变单元大小的方法也是时序优化中最典型的方法之一, 时序违反修复中改变单元大小设置界面如图 9-42 所示。

在时序分析报告中确定需要改变大小的实例化单元, 并填入 Instance 文本框中, 然后在 Options 设置框中选择合适的改变单元大小基本方法。改变单元大小的基本方法有 3 种设置选择, 分别为 upsize、downsize 和 specified cell。

upsized 为使单元尺寸默认只变大一个等级。downsize 为使单元尺寸默认只变小一个等级。specified cell 为使单元尺寸改变为指定的单元大小, 这样变化的单元大小可以存在多个等级的变化。

最后单击 Apply 按钮即可完成改变单元大小修复时序违反的过程。

3. 删除缓冲器单元

删除缓冲器单元的方法也是时序优化中最典型的方法之一，时序违反修复中删除缓冲器单元设置界面如图 9-43 所示。

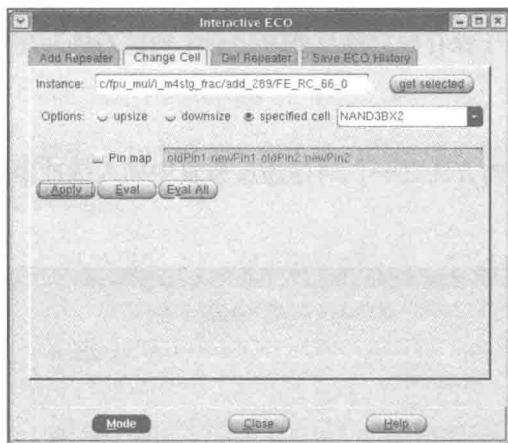


图 9-42 改变单元大小设置界面

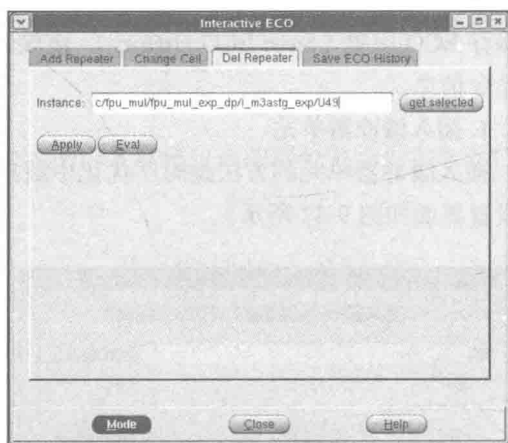


图 9-43 删除缓冲器单元设置界面

通过在时序分析报告中确定需要删除的缓冲器单元，并填入 Instance 文本框中，最后单击 Apply 按钮即可完成改变单元大小修复时序违反的过程。

4. 保存 ECO 历史

在进行时序违反修复过程中，ETS 工具会记录所有使用过的时序违反修复，通过保存 ECO 历史设置可以自动保存时序违反修复过程中所执行的时序优化命令信息。保存 ECO 历史设置界面如图 9-44 所示。

在 ECO HistoryFile 文本框中填入正确的 ECO 输出文件，然后选择文件类型为 ets，最后单击 Save and Close 按钮，即可完成自动保存时序违反修复过程中所执行的时序优化命令信息过程。

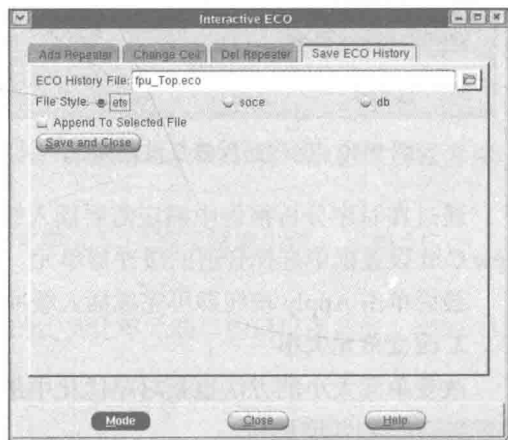


图 9-44 保存 ECO 历史设置界面

第 10 章

Tcl 脚本编程

在静态时序分析中，可以通过编程的方法来达到提高工作效率和质量的目的。时序分析工具都提供 Tcl 编程接口。

Tcl 全称是 Tool command language，它是一种可嵌入的命令脚本化语言，由 John Ousterhout 发明于 1988 年。Tcl 是一个基于字符串的命令语言，数据结构和语法非常简单，代码可重用性高，功能强大并且简单易学，因此经常被用于快速原型开发、脚本编程、GUI 和测试等方面，在信息产业领域具有广泛的应用。

本章只对 Tcl 脚本编程与静态时序分析中经常使用的基本知识进行介绍，对于高阶和更详细的介绍可以参考 Tcl 编程的其他书籍。在本章的例子中，使用 % 为 Tcl 的命令提示符，并在接着的下一行显示命令执行结果。* 后面的内容为注释说明。

10.1 Tcl 语法

Tcl 语法是 Tcl 解释器怎样对 Tcl 命令进行分析的规则的组合。

10.1.1 命令格式

一个 Tcl 脚本可以包含一个或多个命令。命令之间必须用换行符或分号隔开，下面的两个 Tcl 脚本的命令格式都是正确的，如例 10-1 所示。

例 10-1 Tcl 脚本的命令格式

```
%cmd1 a      *cmd1 为 Tcl 命令, a 为变量
%cmd2 b      *cmd2 为 Tcl 命令, b 为变量
或者
%cmd1 a; cmd2 b
```

Tcl 的每一个命令字符串由一个或者几个字符单元组成，命令字符串中开头字符单元代表命令名，其他的字符单元则是命令字符串的参数变量部分，字符单元之间需要用空格隔开进行区分。

Tcl 解释器对一个命令字符串的求值过程分为两个阶段：分析和执行。在分析阶段，Tcl 解释器基于分析规则把命令分成独立的字符单元，同时根据变量定义进行必要的变量替换；在执行阶段，Tcl 解释器会把开头的字符单元当作命令名，并检查这个命令名是否正确，如果该命令名正确，则激活这个命令对应的程序，并把所有的字符单元作为参数变量传递给该命令，通过在执行命令过程中进行参数变量处理完成求值过程。

(1) 输出

输出字符串通过命令 puts 来实现，在 puts 后面的简单文本单元会输出到标准输出设备上，如例 10-2 所示。

| 例 10-2 输出命令 puts | |
|------------------|---------------|
| %puts Hello | * 输出字符串 Hello |
| Hello | |
| %puts World | * 输出字符串 World |
| World | |

如果需要输出多于一个单词的字符串，则需要把这个字符串用双引号或者大括号包含起来，如例 10-3 所示。

| 例 10-3 输出字符串 | |
|----------------------|---------------------|
| %puts "Hello World" | * 输出字符串 Hello world |
| Hello World | |
| %puts {Hello World } | |
| Hello World | |

输出命令 puts 的默认行为是在输出文本结束后输出一个对应换行符 return，可以通过调用参数 -nonewline 选项告诉 puts 命令不输出回车换行。

(2) 赋值

Tcl 的赋值命令是 set，其变量的类型是字符串。set 命令能生成一个变量，也能读取或改变一个变量的值，如例 10-4 所示。

| 例 10-4 赋值命令 | |
|-----------------|----------------------|
| % set a abcdefg | * 定义变量 a 的值为 abcdefg |
| abcdefg | |

例 10-4 中，如果变量 a 没有定义，set 命令将生成变量 a，并将其值置为 abcdefg；如果 a 已定义，就把 a 的值重新设置为 abcdefg。

命令 unset 则与命令 set 作用相反，unset 命令取消变量定义，并释放变量所占的内存空间，并且 unset 命令后面可以对任意多个参数进行操作，每个参数是一个变量名，可以是简单变量，也可以是数组或数组元素，如例 10-5 所示。

例 10-5 删除命令

| | |
|-------------------------------------|--|
| <code>%unset a b c d</code> | * 删除 4 个变量 <code>a</code> 、 <code>b</code> 、 <code>c</code> 、 <code>d</code> |
| <code>% unset ab day(sunday)</code> | * 删除数组 <code>day</code> 的元素 <code>sunday</code> |
| <code>%unset day</code> | * 删除数组 <code>day</code> |

例 10-5 的语句中删除了变量 `a`、`b`、`c`、`d` 和数组元素 `day(sunday)`，如果直接删除数组则只需给出数组的名字，如 `day`。

10.1.2 替换

Tcl 解释器在分析命令时，把所有的命令参数变量都当作字符串看待，不管在内部它可能被表示成列表、整型、浮点或者其他类型，这样可以提高 Tcl 语言的效率。替换操作如例 10-6 所示。

例 10-6 替换操作

| | |
|---------------------------|---|
| <code>%set x 20</code> | * 定义变量 <code>x</code> ，并把 <code>x</code> 的值赋为 20 |
| <code>20</code> | |
| <code>%set y x+200</code> | * <code>y</code> 的值是 <code>x+200</code> ，而不是我们期望的 220 |
| <code>x+200</code> | |

例 10-6 的第二行命令中，`x` 被看作字符串 `x+200` 的一部分，如果想使用 `x` 的值 20，就必须告诉 Tcl 解释器调用变量 `x` 的值，而不是字符 `x` 本身。这就需要使用 Tcl 语言中提供的替换功能。

Tcl 提供 3 种形式的替换：变量替换、命令替换和反斜杠替换。每种替换都会让字符单元被其他的值所代替。替换不仅可以发生在包括命令名在内的每一个字符单元中，而且还可以嵌套。

(1) 变量替换

变量替换功能使用 `$` 符号来实现，变量替换会把变量的值导入字符单元中，如例 10-7 所示。

例 10-7 变量替换

| | |
|-----------------------------|--|
| <code>%set x 20</code> | |
| <code>%set y \$x+200</code> | * <code>y</code> 的值是 20+200，这里变量 <code>x</code> 被替换成值 20 |
| <code>20+200</code> | |

例 10-7 的结果得到 `y` 的值为字符表达式 `20+200`。如果需要通过 TCL 解释器把表达式 `20+200` 的结果值 220 直接赋值给变量 `y`，就必须使用命令替换，这样 TCL 会把 `20+200` 看成一个表达式并求值。

(2) 命令替换

命令替换是由中括号 `[]`、Tcl 命令和其参数组成，命令替换会使某一个命令的所有或部分字符单元被另一个命令的结果所代替，如例 10-8 所示。

例 10-8 命令替换

| | | | |
|--------|---|----------------|------------------------------------|
| %set | x | 20 | |
| 20 | | | |
| %set | y | [expr \$x+200] | * 变量 x 被替换成值 20，并且 y 的值是表达式的结果 220 |
| 20+200 | | | |
| 220 | | | |

例 10-8 的结果 y 的值是 220，当 Tcl 解释器遇到字符 [时，它会把随后的 expr 字符单元作为命令名，并激活与字符单元 expr 匹配的 C/C++ 过程，并把结果替换成命令 expr 和其变量参数所得到的值 220。

命令替换的中括号 [] 内需要一个有效的 Tcl 脚本，长度不限。中括号 [] 内的 Tcl 脚本的值为最后一个命令的返回值，如例 10-9 所示。

例 10-9 中括号 [] 内返回值

| | | | |
|------|---|--------------------------|--------------------------|
| %set | y | [expr \$x+100;set b 300] | * 中括号内最后一个命令为 set b 300 |
| 300 | | | * 其返回值为 300，因此 y 的值为 300 |

通过命令置换，可以完成命令之间的嵌套执行，即一个命令的结果可以作为另外一个命令的参数值。

(3) 反斜杠替换

Tcl 脚本中的反斜杠替换用于在字符单元中插入诸如换行符、空格、[、\$ 等这些被 Tcl 解释器当作特殊符号对待的字符，如例 10-10 所示。

例 10-10 反斜杠替换

| | | | |
|------|---|------|------------|
| %set | x | a\ b | *x 的值为 a b |
| a b | | | |

如果没有 \ 的话，Tcl 会报错，因为解释器会把这里最后两个单词之间的空格认为是分隔符，于是发现 set 命令有多于两个参数，从而报错。加入符号 \ 后，Tcl 解释器就把后面的空格不当作分隔符，a b 被认为是一个字符串单元。

表 10-1 中列出了部分 Tcl 所支持的反斜杠替换。

表 10-1 反斜杠替换命令

| 反斜杠替换命令 | 输出结果 |
|---------|------|
| \a | 铃声 |
| \b | 退格 |
| \f | 清屏 |
| \n | 换行 |
| \r | 回车 |
| \t | 制表 |
| \v | 垂直制表 |

10.1.3 双引号和花括号

除了使用反斜杠外, Tcl 还提供另外两种方法, 使得解释器把分隔符和替换符等特殊字符单元当作普通字符单元, 而不做特殊处理。这就要使用双引号 "" 和大括号 {}, 但是 Tcl 解释器对双引号和大括号的处理还存在不同。

Tcl 解释器对双引号中的各种分隔符将不作处理, 除了对换行符及 \$ 和 [] 两种替换符会按替换操作处理, 如例 10-11 所示。

例 10-11 双引号内特殊字符单元处理

| | |
|------------------|-------------------|
| %set x 20 | * 定义变量 x 值为 20 |
| 20 | |
| %set y "\$x abc" | * 变量 x 仍然被替换成值 20 |
| 20abc | |

而在大括号 {} 中, 所有特殊字符单元都将成为普通字符单元, 失去其特殊意义, Tcl 解释器不会对其做特殊处理, 如例 10-12 所示。

例 10-12 大括号内特殊字符单元处理

| | |
|------------------------------|--------------------|
| %set y {/r\$x [expr 20+200]} | * 大括号 {} 中字符单元保持原样 |
| /n\$x [expr 20+200] | |

10.1.4 注释

编写 Tcl 脚本过程中, 对重要的代码进行注释说明是必要的, Tcl 中的注释符是 #。Tcl 解释器在解析命令中如果遇到第一个字符单元为注释符 # 时, 以后所有的字符单元都被 Tcl 看作注释, Tcl 解释器对注释将不做任何处理。# 必须出现所在命令的第一个, 才被 Tcl 解释器识别为注释, 否则 Tcl 解释器会把 # 当作字符单元解析, 如果不符合 Tcl 语法格式则会报错, 如例 10-13 所示。

例 10-13 注释

| | |
|--|----------------------------------|
| % # comment is here | * 通过字符 # 定义注释行 |
| %set a 500 # comment is here | * 通过字符 # 定义注释行是错误的 |
| wrong # args: should be "set varName ?newValue?" | |
| %set b 501 ; # this is a comment | * Tcl 解释器解析下一条命令时, 因为第一个字符单元 501 |
| | * 为 #, 因此识别为注释行 |

例 10-13 中第二行的字符 # 就不被当作注释符, 因为它出现在命令的中间, Tcl 解释器把它和后面的字符当作命令的参数处理, 从而导致错误。而第四行的字符 # 就被 Tcl 解释器识别为注释, 因为前一个命令已经用一个分号结束, Tcl 解释器解析下一个命令时, 第一个字符为 #, 因此字符 # 就被识别为注释。

10.2 数据结构

Tcl 支持两种类型的数据结构：变量和数组。

10.2.1 简单变量

Tcl 中变量由两个部分组成：变量名字和变量值。变量名字和变量值都可以定义成任意字符串，如例 10-14 所示。

例 10-14 所示定义变量

| | |
|-----------------|----------------------|
| % set a 20 | * 定义变量 a 的值为 20 |
| 20 | |
| %set a.1 40 | * 定义变量 a.1 的值为 40 |
| 40 | |
| % set b \${a.1} | * 定义变量 b 的值为 \${a.1} |
| 20.1 | |

例 10-14 中最后一个命令行中，Tcl 解释器只把 \$ 符号之后直到第一个不是字母、数字或下划线的字符（这里是 '!'）之间的单词符号（这里是 'a'）当作要被替换变量的名字，因此 Tcl 解释器把 a 替换成 20，然后把字符串“20.1”赋值给变量 b。如果初衷是把变量 a.1 的值付给 b，根据前面介绍的特殊字符单元处理，需要用大括号把变量名括起来，如例 10-15 所示。

例 10-15 定义特殊变量

| | |
|----------------|----------------------|
| %set a.1 40 | * 定义变量 a.1 的值为 40 |
| 40 | |
| %set b \${a.1} | * 定义变量 b 的值为 \${a.1} |
| 40 | |

10.2.2 数组

数组是一些元素的集合，Tcl 中定义一个数组需要连数组元素一起定义。因此在数组中，数组元素的变量名包含两部分：数组名和数组元素的名字。Tcl 中数组元素的名字（下标）可以定义为任何字符串，如例 10-16 所示。

例 10-16 定义数组变量

| | | | |
|------|-------------|---|----------------------------|
| %set | day(sunday) | 7 | * 定义数组变量 day(sunday) 的值为 7 |
| %set | day(may) | 5 | * 定义数组变量 day(may) 的值为 5 |

例 10-16 中第一个命令定义名为 day 的数组，同时在数组中生成一个名为 sunday 的数组元素，并把值置为 7，第二个命令生成一个名为 may 的数组元素，并把值设置为 5。

10.3 表达式

Tcl 中的表达式类似于 ANSI C 的表达式。表达式由操作数和操作符构成，下面分别介绍。

10.3.1 操作数

Tcl 表达式的操作数通常是整数 (int)、浮点数 (float) 和字符串 (string)。整数一般是十进制的, 但如果整数的第一个字符是 0 (zero), 那么 Tcl 将把这个整数看作八进制的, 如果前两个字符是 0x 则这个整数被看作是十六进制的。Tcl 中实数的写法与 ANSI C 中完全一样, 如例 10-17 所示。

例 10-17 操作数举例

| | |
|---------|------------|
| 5.1 | * 该操作数为浮点数 |
| 7.3e+12 | * 该操作数为浮点数 |
| 8 | * 该操作数为整数 |
| Hello | * 该操作数为字符串 |

10.3.2 运算符和优先级

表 10-2 中列出了 Tcl 中用到的运算符, 它们的语法形式和用法与 ANSI C 中很相似, 这里不一一介绍。表 10-2 中的运算符是按优先级从高到低往下排列的, 同一格中的运算符优先级相同。

表 10-2 运算符和优先级

| 语法形式 | 结 果 | 操作数类型 |
|-------|------------|------------------|
| -a | 负 <i>a</i> | int,float |
| a*b | 乘 | int,float |
| a+b | 加 | int,float |
| a<<b | 左移位 | int |
| a<b | 小于 | int,float,string |
| a==b | 等于 | int,float,string |
| a&b | 位操作与 | int |
| a^b | 位操作异或 | int |
| a b | 位操作或 | int |
| a&& b | 逻辑与 | int,float |
| a b | 逻辑或 | int,float |
| a?b:c | 选择运算 | int,float |
| a!=b | 不等于 | int,float,string |
| a>>b | 右移位 | int |
| a-b | 减 | int,float |
| !a | 非 <i>a</i> | int,float |
| ~a | 按位取反 | int |
| a/b | 除 | int,float |
| a%b | 取模 | int |
| a>b | 大于 | int,float,string |
| a<=b | 小于等于 | int,float,string |
| a>=b | 大于等于 | int,float,string |

10.3.3 数学函数

Tcl 支持常用的数学函数，表达式中数学函数的写法类似于 C/C++ 语言的写法，数学函数的参数可以是任意表达式，多个参数之间用逗号隔开，如例 10-18 所示。

例 10-18 数学函数

| | |
|--------------|-----------------------|
| %set x 3 | * 定义变量 x 的值为 3 |
| 3 | |
| % expr 2*\$x | * 定义数学函数表达式 2*\$x 并求值 |
| 6 | |

例 10-18 中通过表达式中的数学函数进行求值。需要注意的一点是，数学函数并不是命令，只有在表达式中出现才有意义。

Tcl 中支持的主要数学函数如表 10-3 所示。

表 10-3 主要数学函数

| 数学函数名 | 说 明 |
|-------------|-------------------------------|
| abs(x) | 取绝对值 |
| acos(x) | 反余弦 |
| asin(x) | 反正弦 |
| atan(x) | 反正切 |
| atan2(x, y) | 比值取反正切 |
| ceil(x) | 返回不小于 x 值的最小整数 |
| cos(x) | 余弦 |
| cosh(x) | 双曲余弦 |
| double(i) | 转换双精度 |
| exp(x) | e 为底的指数函数 |
| floor(x) | 返回不大于 x 值的最大整数 |
| fmod(x, y) | 浮点数取模（求余） |
| hypot(x, y) | 根据直角三角形的两直角边长度 x 和 y 计算其斜边的长度 |
| int(x) | 取整 |
| log(x) | 自然对数 |
| log10(x) | 以 10 为底的对数 |
| pow(x, y) | 幂运算 |
| round(x) | 四舍五入取整数 |
| sin(x) | 正弦 |
| sinh(x) | 双曲正弦 |
| sqrt(x) | 求二次根 |
| tan(x) | 正切 |
| tanh(x) | 三角函数 |
| rand() | 取 0 ~ 1 之间的随机实数（无输入参数） |
| srand(x) | 以整数 arg 为随机数生成器的种子产生随机数 |

Tcl 中有很多命令都以表达式作为参数。最典型的是 `expr` 命令，另外 `if`、`while`、`for` 等循环控制命令的循环控制中也使用表达式作为参数。

`expr` 是 Tcl 的一个命令，语法为：

```
expr arg ?arg ...?
```

这里两个 `?` 之间的参数表示可以省略，后面介绍命令时对于可省略的参数都使用这种表示形式。`expr` 可以有一个或多个参数，它把所有的参数组合到一起，作为一个表达式，然后求值，如例 10-19 所示。

例 10-19 `expr` 命令求值

| | |
|----------------------------|------------------------|
| <code>%expr 1+3*3</code> | * 数学表达式进行求值 |
| <code>10</code> | |
| <code>%expr abs(-3)</code> | * 对 -3 进行求绝对值操作并返回结果 3 |
| <code>3</code> | |

10.3.4 列表集合

Tcl 中 `list` 表示列表集合，它是由一堆元素组成的有序列表集合。`list` 可以嵌套定义，`list` 每个元素可以是任意字符串，也可以是 `list`。下面都是 Tcl 中的合法的 `list`，如例 10-20 所示。

例 10-20 `list` 用法

| | |
|--------------------------|------------------|
| <code>{}</code> | * 一个没有元素的空列表集合 |
| <code>{a b c d}</code> | * 包含 4 个元素的列表集合 |
| <code>{a {b c} d}</code> | * 含有嵌套子列表集合的列表集合 |

`list` 是 Tcl 中比较重要的一种数据结构，对于编写复杂的脚本有很大的帮助。Tcl 提供了很多基本命令对集合进行操作，下面一一介绍。

1. `list` 命令

语法：

```
list ? value value...?
```

`list` 命令定义一个列表集合，`list` 的元素就是所有的值，如例 10-21 所示。

例 10-21 `list` 命令

| | |
|--|-----------------------------------|
| <code>% list 1 2 {3 4}</code> | * 通过 <code>list</code> 命令定义一个列表集合 |
| <code>1 2 {3 4}</code> | |
| <code>% set a [list Sun Mon Tues]</code> | * 定义一个列表集合名为 <code>a</code> 的列表集合 |
| <code>Sun Mon Tues</code> | |

2. `concat` 命令

语法：

```
concat list ?list...?
```

concat 命令把多个 list 合成一个 list，每个 list 变成新 list 的一个元素，如例 10-22 所示。

例 10-22 concat 命令

| | |
|-------------------------|---|
| %set a [list 1 2 3] | * 定义列表集合名为 a 的列表集合 |
| 1 2 3 | |
| %set b [list 4 5 6] | * 定义列表集合名为 b 的列表集合 |
| 4 5 6 | |
| %set y [concat \$a \$b] | * 定义集合名为 y 的列表集合，其元素为列表集合 a 和列表集合 b 的总和 |
| 1 2 3 4 5 6 | |

3. lindex 命令

语法：

```
lindex list index
```

lindex 命令返回 list 的第 index 个元素，其中初始值从 0 开始，如例 10-23 所示。

例 10-23 lindex 命令

| | |
|------------------------|----------------|
| % lindex {1 2 {3 4}} 1 | * 返回第 1 个元素为 2 |
| 2 | |

4. llength 命令

语法：

```
llength list
```

llength 命令返回 list 的元素个数，如例 10-24 所示。

例 10-24 llength 命令

| | |
|-----------------------|--------------|
| % llength {5 2 {8 9}} | * 返回列表集合的长度值 |
| 3 | |

5. linsert 命令

语法：

```
linsert list index value ?value...?
```

linsert 命返回一个新串，新串把所有的 value 参数值插入 list 的第 index 个（0-based）元素之前得到，如例 10-25 所示。

例 10-25 linsert 命令

| | |
|-----------------------------------|----------------------------|
| % linsert {1 2 {3 4}} 1 3 5 {1 1} | * 把列表集合 {3 5 {1 1}} 插入列表集合 |
| | * {1 2 {3 4}} 的第 |
| 1 3 5 {1 1} 2 {3 4} | 1 位 |

6. lreplace 命令

语法:

```
lreplace list first last ?value value ...?
```

lreplace 命令返回一个新串, 新串是把 list 的第 first 到第 last 个元素用所有的 value 参数替换得到的。如果没有 value 参数, 就表示删除第 first 到第 last 个元素, 如例 10-26 所示。

例 10-26 lreplace 命令

```
% lreplace {2 7 8 {1 11} 2 {13 14}} 3 3 * 把旧列表集合中的第 3 个元素删除
2 7 8 2 {13 14}
% lreplace {1 7 8 2 {5 6}} 4 4 7 8 9 * 把旧列表集合中的第 4 个元素替换成 7 8 9
1 7 8 2 7 8 9
```

7. lrange 命令

语法:

```
lrange list first last
```

lrange 命令返回 list 的第 first 到第 last 元素组成的集合串, 如果 last 的值是 end, 就是从第 first 个直到串的最后, 如例 10-27 所示。

例 10-27 lrange 命令

```
% lrange {1 7 8 9 7 8 6} 3 end * 返回列表结合的第 3 个到最后的所有元素
9 7 8 6
```

8. lappend 命令

语法:

```
lappend varname value ?value...?
```

lappend 命令把每个 value 的值作为一个元素附加到变量 varname 后面, 并返回变量的新值, 如果 varname 不存在, 就生成这个变量, 如例 10-28 所示。

例 10-28 lappend 命令

```
% lappend a 4 6 5 * 把集合 4 6 5 附加到变量 a 中
4 6 5
% set a * 变量 a 的值为 4 6 5
4 6 5
```

9. lsearch 命令

语法:

```
lsearch ?-exact? ?-glob? ?-regexp? list pattern
```

lsearch 命令返回 list 中第一个匹配模式 pattern 的元素的索引, 如果找不到匹配就返

回 -1。-exact、-glob、-regexp 是 3 种模式匹配的技术。-exact 表示精确匹配；-glob 的匹配方式和 string match 命令的匹配方式相同，将在后面介绍 string 命令时详细讲解；-regexp 表示正则表达式匹配，将在后面介绍 regexp 命令时详细讲解。lsearch 支持通配符 * 格式，但可以使用 -exact 选项将其屏蔽而进行精确匹配，缺省时使用 -glob 匹配，如例 10-29 所示。

例 10-29 lsearch 命令

```
% set a { a b cd }      * 定义变量 a 的值为列表集合 a b cd
a b cd
% lsearch $a c*          * 变量 a 中匹配 c* 的元素为字符串中的第 2 个
2
% lsearch $a c?          * 变量 a 中没有匹配 c? 的元素
-1
```

10. lsort 命令

语法：

```
lsort ?options? list
```

lsort 命令用于返回把 list 进行排序后的列表集合串。

options 可以使用如下值：

-ascii 按 ASCII 字符的顺序排序比较，这是缺省情况。

-dictionary 按字典排序，与 -ascii 不同的地方是：

□ 不考虑大小写。

□ 如果元素中有数字的话，数字被当作整数来排序。

bigBoy 排在 bigbang 和 bigboy 之间，x10y 排在 x9y 和 x11y 之间。

-integer 把 list 的元素转换成整数，按整数排序。

-real 把 list 的元素转换成浮点数，按浮点数排序。

-increasing 升序（按 ASCII 字符比较）

-decreasing 降序（按 ASCII 字符比较）

-command command TCL 自动利用 command 命令把每两个元素一一比较，然后给出排序结果，如例 10-30 所示。

例 10-30 lsort 命令

```
%set a {3 1 4 2}      * 定义变量 a 的值为列表集合 13 1 4 1.2
%lsort -increasing $a  * 按 ASCII 字符值大小由小到大对列表集合 a 进行重新排列
1 2 3 4
```

11. split 命令

语法：

```
split string ?splitChars?
```

`split` 命令把字符串 `string` 按分隔符 `splitChars` 分成一个个单独的字符单元，返回由这些字符单元组成的串。如果 `splitChars` 是一个空字符 `{}`，`string` 被按字符分开。如果 `splitChars` 没有给出，以空格为分隔符，如例 10-31 所示。

例 10-31 `split` 命令

| | |
|-----------------------------------|-----------------------------------|
| <code>% split " a.b.c " .</code> | * 根据指定的分隔符，来分隔字符串 |
| <code>a b c</code> | |
| <code>% split " a b cd "</code> | * 根据默认分隔符空格来分隔字符串 |
| <code>a b cd</code> | |
| <code>% split " a b cd" {}</code> | * 根据分隔符 <code>{}</code> 按字符来分隔字符串 |
| <code>a { } b { } c d</code> | |

12. `join` 命令

语法：

```
join list ?joinString?
```

`join` 命令是命令 `split` 的反向操作。这个命令把 `list` 的所有元素合并到一个字符串中，中间加入分隔符 `joinString` 来分开字符串。缺省的 `joinString` 是空格，如例 10-32 所示。

例 10-32 `join` 命令

| | |
|--|----------------------------------|
| <code>% join { a { } b { } c d } {}</code> | * 加入指定分隔符 <code>{}</code> 来合并字符串 |
| <code>a b cd</code> | |
| <code>% join { a b c d }.</code> | * 加入指定分隔符，来合并字符串 |
| <code>a.b.c.d</code> | |

10.4 控制流

Tcl 中控制结构允许程序根据不同的状态、条件和参数来选择不同的处理和执行路径，从而使代码具有更强的灵活性、健壮性和可读性。Tcl 中的控制流和 C 语言类似，包括 `if`、`while`、`for`、`foreach`、`switch`、`break`、`continue` 等命令。下面将对这些控制流命令进行详细介绍。

10.4.1 `if` 命令

语法：

```
if test1 body1 ?elseif test2 body2 elseif.... ? ?else bodyn?
```

`if` 命令先把 `test1` 当作一个表达式求值，如果值非 0，则把 `body1` 当作一个脚本执行并返回所得值，否则把 `test2` 当作一个表达式求值，如果值非 0，则把 `body2` 当作一个脚本执行并返回所得值，以此类推。如例 10-33 所示。

例 10-33 if 命令

| | |
|-------------------------|----------------|
| %if { \$x>1 } { | * 第一个 if 循环判断 |
| | |
| }elseif{ \$x == 10 } { | * 第二个 if 循环判断 |
| | |
| }elseif { \$x == 12 } { | * 第三个 if 循环判断 |
| | |
| }else{ | * 最后一个 if 循环判断 |
| | |
| } | |

例 10-33 中的大括号“{”一定要和 if 命令在同一行上，因为如果不这样做，Tcl 解释器会认为 if 命令在换行符处已结束，下一行会被当成新的命令，从而导致错误的结果。在接下来介绍的循环命令的书写中也要注意这个问题。书写中还要注意的一个问题是 if 和大括号之间应该有一个空格，否则 Tcl 解释器会把“if{”作为一个整体，当作一个命令名，从而导致错误。

10.4.2 循环命令

循环命令主要包括包括 while、for、foreach 等，下面将进行一一介绍。

1. while 命令

语法：

```
while test body
```

while 命令中参数 test 是一个表达式，body 是一个脚本，如果表达式的值非 0，就运行脚本，直到表达式为 0 才停止循环，此时 while 命令中断并返回一个空字符串，如例 10-34 所示。

例 10-34 while 循环命令

| | |
|--------------------------------|---------------------------|
| %set a {1 2 3 4 5 6 7 8} | * 设置变量 a |
| %set b " " | * 设置变量 b |
| %set i [expr [llength \$a] -1] | * 得到变量 a 的元素个数 |
| %while { \$i>=0}{ | * 进入 while 循环判断，并运行循环中的脚本 |
| lappend b [lindex \$a \$i] | |
| incr i -1 | |
| } | |

例 10-34 中设置变量 a 是一个列表集合，下面的脚本通过 while 循环命令把变量 a 的值复制到变量 b，直到 while 循环条件表达式的值为 0 时跳出 while 循环。

2. for 命令

语法：

```
for init test reinit body
```

for 命令中参数 init 是一个初始化脚本，第二个参数 test 是一个表达式，用来决定循环什么时候中断，第三个参数 reinit 是一个重新初始化的脚本，第四个参数 body 也是脚本，代表循环体。for 循环需要预置条件或者初始化命令，告诉 for 命令起始执行条件，如例 10-35 所示。

例 10-35 for 循环命令

| | |
|---|----------------|
| %set a {1 2 3 4 5 6 7 8} | * 设置变量 a |
| %set b " " | * 设置变量 b |
| %for {set i [expr [llength \$a] -1]} {\$i>=0} {incr i -1} { | * 进入 for 循环，并运 |
| lappend b [lindex \$a \$i] } | * 行循环中的脚本 |

例 10-35 中设置变量 *a* 是一个列表集合，下面的脚本通过 for 循环命令把变量 *a* 的值复制到变量 *b*，直到 for 完成预置条件后跳出循环。

3. foreach 命令

foreach 命令会遍历整个列表，依次取出列表的每个元素的值放到指定变量中。foreach 命令有两种语法形式。

(1) foreach varName list body

foreach 命令用法 1 中的第一个参数 varName 是一个变量，第二个参数 list 是一个表（有序集合），第三个参数 body 是循环体。每次取得链表的一个元素，都会执行循环体一次，如例 10-36 所示。

例 10-36 foreach 命令用法 1

| | |
|---------------------------|---------------------|
| %set a {1 2 3 4 5 6 7 8} | * 设置变量 a |
| %set b " " | * 设置变量 b |
| %foreach i \$a{ | * 将变量 a 的元素加入变量 b 中 |
| set b [linsert \$b 0 \$i] | |
| } | |

(2) foreach varlist1 list1 ?varlist2 list2 ...? Body

foreach 命令用法 2 的这种形式包含了第一种用法。第一个参数 varlist1 是一个循环变量列表，第二个参数是一个列表 list1，varlist1 中的变量会分别取 list1 中的值。body 参数是循环体。?varlist2 list2 ...? 表示可以有多个变量列表和列表对出现，如例 10-37 所示。

例 10-37 foreach 命令用法 2

| | |
|----------------------------------|---|
| %set x {} | * 设置变量 x |
| %foreach {i j} {a b c d e f} { | |
| lappend x \$j \$i | |
| } | * 该 foreach 循环总共有三次循环，x 的值为 "b a d c f e" |
| | |
| %set x {} | |
| %foreach i {a b c} j {d e f g} { | |

```
lappend x $i $j
}          * 该 foreach 循环总共有四次循环, x 的值为 "a d b e c f {} g"

%set x {}
%foreach i {a b c} {j k} {d e f g} {
lappend x $i $j $k
}          * 该 foreach 总共有三次循环, x 的值为 "a d e b f g c {} {}"
```

(3) break 和 continue 命令

在循环体中, 可以用 break 和 continue 命令中断循环。其中 break 命令结束整个循环过程, 并从循环中跳出, 如例 10-38 所示。

例 10-38 break 命令

```
%set x {} * 设置变量 x
%foreach {i j} {a b c d e f} {
lappend x $j $i
break      * 该 foreach 循环默认有 3 次循环, 但是由于有 break 命令, 只循环一次就跳出循环
}
```

continue 只是结束本次循环, 如例 10-39 所示。

例 10-39 continue 命令

```
%set x {} * 设置变量 x
%foreach {i j} {a b c d e f} {
if {$x == "a b"} {
continue   * 该 foreach 循环默认共有三次循环, 但是由于有 continue 命令,
            * 只要不满足 if 条件即跳出本次循环
}
lappend x $j $i
}
```

(4) switch 命令

语法:

```
switch ? options? string { pattern body ? pattern body ...?}
```

switch 命令的第一个是可选参数 options, 表示进行匹配的方式。Tcl 支持 3 种匹配方式: -exact 方式, -glob 方式, -regexp 方式, 缺省情况表示 -glob 方式。-exact 方式表示的是精确匹配。第二个参数 string 是要被用来作测试的值, 第三个参数是括起来的一个或多个元素对, 如例 10-40 所示。

例 10-40 switch 命令

```
%switch $x {
a      -
b      {incr t1}
```

```
c {incr t2}
default {incr t3}
}
```

例 10-40 中 a 的后面跟一个 '!' 表示使用和下一个模式相同的脚本。default 表示匹配任意值。一旦 switch 命令找到一个模式匹配, 就执行相应的脚本, 并返回脚本的值, 作为 switch 命令的返回值。

和 C 语言中 switch 语句一样, Tcl 中的 switch 命令也可以由 if 命令实现, 只是书写起来较为繁琐。

10.5 eval 命令

语法:

```
eval arg ?arg ...?
```

eval 命令是一个用来构造和执行 Tcl 脚本的命令, 它可以接收一个或多个参数, 然后把所有的参数以空格隔开组合到一起成为一个脚本, 然后对这个脚本进行求值, 如例 10-41 所示。

例 10-41 eval 命令

```
% eval set a 2 ;set b 4
4
```

eval 命令按照 concat 风格将输入参数连接成命令字符串, 然后调用 Tcl_Eval 来完成命令计算和执行。

10.6 source 命令

source 命令读一个文件并把这个文件的内容作为一个脚本进行求值, 如例 10-42 所示。

例 10-42 source 命令

```
%source hello.Tcl * 运行 hello.Tcl 脚本
```

10.7 过程

Tcl 支持过程的定义和调用, 在 Tcl 中, 过程可以看作是用 Tcl 脚本实现的命令, 一个过程可以像 Tcl 内建命令一样直接使用, 效果与 Tcl 的固有命令相似。过程可以有自己的内部变量, 而且起作用的范围仅限过程内部, 在过程外部无法使用、获取这些变量的值。这就引

出了作用域、局部变量和全局变量的概念。我们可以在任何时候使用 `proc` 命令定义自己的过程，Tcl 中的过程类似于 C 中的函数。

10.7.1 过程定义和返回值

Tcl 中过程是由 `proc` 命令产生的，语法如下：

```
proc procName { var1 var2 ... } {
    body ...
}
```

`proc` 命令有 3 个参数：`procName` 是定义的过程名字；`{var1 var2 ...}` 是输入、输出参数列表；`body` 是过程执行命令体。`body` 的界定大括弧和 `if` 等命令的命令执行体遵循相同的规则和注意事项。可以使用 `return` 命令在需要的时候返回调用程序，如例 10-43 所示。

例 10-43 `proc` 命令

```
% proc add {x y} {expr $x+$y}    * 定义名为 add 的过程程序
```

`proc` 命令的第一个参数是要定义的过程名字 `add`，第二个参数是过程的参数列表 `{x y}`，参数之间用空格隔开，第三个参数是一个 Tcl 脚本，代表过程程序体。`proc` 生成一个新的命令，可以像固有命令一样调用，如例 10-44 所示。

例 10-44 调用 `proc` 命令

```
% add 1 5                        * 调用名为 add 的过程程序
6
```

在定义过程时，可以利用 `return` 命令在任何地方返回你想要的值。`return` 命令迅速中断过程，并把它的参数作为过程的结果，如例 10-45 所示。

例 10-45 `return` 命令

```
% proc abs {x} {
    if {$x >= 0} { return $x }
    return [expr -$x]
}
```

过程的返回值是过程体中最后执行的那条命令的返回值。

10.7.2 局部变量和全局变量

对于在过程中定义的变量中分为局部变量和全局变量两种：当变量只能在过程中被访问，并且在过程退出后会被自动删除时，称为局部变量；在所有过程之外定义的变量称之为全局变量。Tcl 中，局部变量和全局变量可以同名，两者的作用域的交集为空，局部变量的作用域是它所在的过程的内部；全局变量的作用域则不包括所有过程的内部。如果我们在过程

内部引用一个全局变量的值，可以使用 `global` 命令，如例 10-46 所示。

例 10-46 全局变量用法

| | |
|------------------------------------|------------------------------|
| <code>% set a 4</code> | * 定义变量 a |
| <code>4</code> | |
| <code>% proc sample { x } {</code> | * 定义名为 sample 的过程 |
| <code>global a</code> | * 调用全局变量 a |
| <code>incr a</code> | |
| <code>return [expr \$a+\$x]</code> | |
| <code>}</code> | |
| <code>% sample 3</code> | * 调用名为 sample 的过程程序 |
| <code>8</code> | |
| <code>%set a</code> | * 全局变量 a 的值在调用 sample 过程时被修改 |
| <code>5</code> | |

例 10-46 中的全局变量 `a` 在过程中被访问并修改，这样在过程中对 `a` 的改变会直接反映到全局上。如果去掉语句 `global a`，Tcl 会出错，因为它不认识变量 `a`。

还有另外一种方式来直接使用全局变量，即用双冒号 `::`，如例 10-47 所示。

例 10-47 双冒号 (::) 全局变量

| | |
|--------------------------------------|--------------------|
| <code>% set a 4</code> | * 定义变量 a |
| <code>4</code> | |
| <code>% proc sample { x } {</code> | * 定义名为 sample 的过程 |
| <code>return [expr \$::a+\$x]</code> | |
| <code>}</code> | |
| <code>% sample 3</code> | * 调用名为 sample 过程程序 |
| <code>7</code> | |

例 10-47 中通过双冒号 `::` 调用全局变量的方式，可以避免过程中局部变量和所要使用的全局变量同名的问题，也可以使代码更清晰。

10.7.3 默认参数和可变个数参数

使用过程的时候，不一定输入所有的参数值。过程的输入参数可以有默认值。默认值由 { 默认参数名默认值 } 指定。如果调用过程时没有指定这些参数的值，则会使用其默认值，否则使用输入值来替代默认值。在使用默认参数的时候要注意，如果默认参数之后还有非默认参数，则在调用此过程的时候，默认参数的值也要求输入，否则会出错。这是因为 Tcl 调用过程的时候是根据位置来匹配参数和输入值的。

Tcl 在过程定义中还提供 3 种特殊的参数形式：

1) 可以定义一个没有参数的过程，如例 10-48 所示。

例 10-48 无参数的过程

| | |
|---------------------------------------|--------------|
| <code>%proc add {} { expr 2+3}</code> | * 过程中的参数列表为空 |
|---------------------------------------|--------------|

2) 可以定义具有默认参数值的过程，即可以为过程的部分或全部参数提供默认值，如果调用过程时未提供那些参数的值，那么过程会自动将默认值赋给相应的参数。和 C/C++ 中具有默认参数值的函数一样，有默认值的参数只能位于参数列表的后部，即在第一个具有默认值的参数后面的所有参数，都只能是具有默认值的参数，如例 10-49 所示。

例 10-49 默认参数值的过程

```
%proc add {val1 {val2 2} {val3 3}}{ * 为过程定义默认值
    expr $val1+$val2+$val3
}

%add 1 * 调用过程时需要使用默认值
6
%add 2 20 * 调用过程时需要使用默认值
25
%add 4 5 6 * 调用过程时不需要使用默认值
15
```

3) Tcl 的过程定义还支持可变个数的参数，如果过程的最后一个参数是 args，那么就表示这个过程支持可变个数的参数调用。调用时，位于 args 以前的参数象普通参数一样处理，但任何附加的参数都需要在过程体中做特殊处理，过程的局部变量 args 将会被设置为一个列表，其元素就是所有附加的变量。如果没有附加的变量，args 就设置成一个空串，如例 10-50 所示。

例 10-50 可变个数参数的过程

```
%proc add { val1 args } { * 为过程定义可变个数参数
    set sum $val1
    foreach i $args {
        incr sum $i
    }
    return $sum
}

%add 2 * 定义了可变个数参数的过程的执行结果
2
%add 2 3 4 5 6
20
```

10.8 引用

语法：

```
upvar ?level? otherVar myVar ?otherVar myVar ...?
```

upvar 命令使得用户可以在过程中对全局变量或其他过程中的局部变量进行访问。upvar 命令的第一个参数 otherVar 是希望以引用方式访问的参数的名字，第二个参数 myVar 是这个

过程中的局部变量的名字，一旦使用 `upvar` 命令把 `otherVar` 和 `myVar` 绑定，那么在过程中对局部变量 `myVar` 的读写就相当于对这个过程的调用者中 `otherVar` 所代表的局部变量的读写，如例 10-51 所示。

例 10-51 `upvar` 命令

```
% proc temp { arg } {
    upvar $arg b          * 在过程中定义引用
    set b [expr $b+2]
}
% proc myexp { var } {
    set a 4
    temp a                * 调用有定义引用的过程程序
    return [expr $var+$a]
}
% myexp 7
13
```

在例 10-51 中，`upvar` 把 `$arg`（实际上是过程 `myexp` 中的变量 `a`）和过程 `temp` 中的变量 `b` 绑定，对 `b` 的读写就相当于对 `a` 的读写。

一个过程 A 或可以被另一个过程 B 所调用，而过程 B 也可被过程 C 调用，那么 A、B、C 三个过程组成了一个调用层次，A 为最低，C 为较高层（`global` 全局过程为最高）。则 A 就可以通过 `upvar` 命令来引用过程 B、C 和全局过程的变量，而不像 `global` 命令只能使用全局变量。如果 `level` 为 1（默认），则表示引用相邻上一层（过程 B）的参数，如果为 2，则表示引用的是向上第 2 层（过程 C）的参数。

`upvar` 命令语法中的 `level` 参数表示调用 `upvar` 命令的过程相对于我们希望引用的变量 `myVar` 在调用栈中相对位置，如例 10-52 所示。

例 10-52 `level` 参数用法

```
upvar 2 other x
```

例 10-52 中，`upvar` 命令使得当前过程调用者的调用者中的变量 `other`，可以在当前过程中利用 `x` 访问。默认情况下，`level` 的值为 1，即当前过程（例 10-51 中的 `temp`）的调用者（例 10-51 中的 `myexp`）中的变量（例 10-51 中 `myexp` 的 `a`）可以在当前过程中利用局部变量（例 10-51 中 `temp` 的 `b`）访问。

如果要访问全局变量可以这样写，如例 10-53 所示。

例 10-53 访问全局变量定义

```
upvar #0 other x
```

例 10-53 中不管当前过程处于调用栈中的什么位置，都可以在当前过程中利用 `x` 访问全局变量 `other`。

10.9 字符串操作

由于 Tcl 程序中把所有的输入都当作字符串看待，所以 Tcl 提供了较强的字符串操作功能，Tcl 中与字符串操作有关的命令有：string、format、regexp、regsub、scan 等。

1. format 命令

语法：

```
format formatstring ?value value...?
```

format 命令类似于 ANSI C 中的 sprintf 函数和 MFC 中 CString 类提供的 Format 成员函数。它按 formatstring 提供的格式，把各个 value 的值组合到 formatstring 中，形成一个被格式化的新字符串，并返回。

formatstring 包含了格式说明关键词和附加文字，使用 % 来引入一个关键词，后跟 0 个或者多个修饰符，然后使用一个转换格式符结尾。value 的参数设置主要有 6 种：位置说明符、标志、字段宽度、精度、长度和格式转换符。

主要的格式转换符在表 10-4 一一列出，并给予了说明。

表 10-4 转换符说明

| 转换符 | 说 明 |
|--------|--|
| d | 有符号整数格式 |
| u | 无符号整数格式 |
| i | 有符号整数格式。可以是十六进制 (0x) 或八进制 (0) 格式 |
| o | 无符号八进制数格式 |
| x 或者 X | 无符号十六进制数格式 |
| c | 将整数映射到对应的 ASCII 字符格式 |
| s | 字符串格式 |
| f | 浮点数格式 |
| e 或者 E | 用科学记数法表示的浮点数格式 |
| g 或者 G | 以 %f 或 %e 格式 (要短一些) 来表示的浮点数格式 |
| - | 字段左对齐 |
| + | 字段右对齐 |
| space | 数字前加一个空格，除非数字带有前导符号 |
| 0 | 使用 0 作为补白 |
| # | 0 表示八进制，0x 表示十六进制数。浮点数中总要带上小数点。不删除末尾的 0 (%g) 尾的 0 (%g) |

此命令不会改变被操作字符串的内容，如例 10-54 所示。

例 10-54 format 命令

| | |
|-------------------|-------------|
| %set name DreamXu | * 定义变量 name |
| DreamXu | |
| %set age 34 | * 定义变量 age |

```
34
%set msg [format "%s is %d years old" $name $age]
DreamXu is 34 years old
```

2. scan 命令

语法:

```
scan string format varName ?varName ...?
```

scan 命令可以认为是 format 命令的反向操作，其功能类似于 ANSI C 中的 sscanf 函数。它按 format 提供的格式分析 string 字符串，然后把结果存到变量 varName 中，注意除了空格和 TAB 键之外，string 和 format 中的字符和“%”必须匹配，如例 10-55 所示。

例 10-55 scan 命令

```
% scan "some 26 34" "some %d %d" a b      * 调用 scan 命令
2
% set a                                     * 变量 a 的值
26
% set b                                     * 变量 b 的值
34
% scan "12.34.56.78" "%d.%d.%d.%d" c d e f
4
% puts [format "the value of c is %d,d is %d,e is %d ,f is %d" $c $d $e $f]
the value of c is 12,d is 34,e is 56 ,f is 78
```

scan 命令的返回值是匹配的变量个数。而且，如果变量 varName 不存在的话，Tcl 会自动声明该变量。

3. regexp 命令

语法:

```
regexp ?switchs? ?--? exp string ?matchVar? \ ?subMatchVar subMatchVar...?
```

regexp 命令用于判断正则表达式 exp 是否全部或部分匹配字符串 string，匹配返回 1，否则 0。

在正则表达式中，一些字符具有特殊的含义，表 10-5 一一列出，并给予了解释。

表 10-5 字符具有特殊的含义

| 字 符 | 意 义 |
|---------|---|
| . | 匹配任意单个字符 |
| ^ | 表示从头进行匹配 |
| \$ | 表示从末尾进行匹配 |
| \x | 匹配字符 x，这可以抑制字符 x 的含义 |
| [chars] | 匹配字符集合 chars 中给出的任意字符，如果 chars 中的第一个字符是 ^，表示匹配任意不在 chars 中的字符，chars 的表示方法支持 a-z 之类的表示 |

(续)

| 字 符 | 意 义 |
|-----------------|---------------------------|
| (regexp) | 把 regexp 作为一个单项进行匹配 |
| * | 对 * 前面的项 0 进行次或多次匹配 |
| + | 对 + 前面的项进行 1 次或多次匹配 |
| ? | 对 ? 前面的项进行 0 次或 1 次匹配 |
| regexp1 regexp2 | 匹配 regexp1 或 regexp2 中的一项 |

通过下面的一个正则表达式进行说明：

```
^((0x)?[0-9a-fA-F]+|[0-9]+)$
```

这里存在两个正则表达式匹配任何十六进制或十进制的整数。两个正则表达式以 | 分开 (0x)? [0-9a-fA-F]+ 和 [0-9]+, 表示可以匹配其中的任何一个。事实上前者匹配十六进制, 后者匹配的十进制。^ 表示必须从头进行匹配, 从而上述正则表达式不匹配 jk12 之类不是以 0x 或数字开头的串。\$ 表示必须从末尾开始匹配, 从而上述正则表达式不匹配 12jk 之类不是数字或 a-fA-F 结尾的串。

正则表达式 (0x)? [0-9a-fA-F]+ 中, (0x) 表示 0x 一起作为一项, ? 表示前一项 (0x) 可以出现 0 次或多次, [0-9a-fA-F] 表示可以是任意 0 ~ 9 之间的单个数字或 a ~ f 或 A ~ F 之间的单个字母, “+” 表示像前面那样的单个数字或字母可以重复出现一次或多次, 如例 10-56 所示。

例 10-56 regexp 命令

```
% regexp {^((0x)?[0-9a-fA-F]+|[0-9]+)$} ab
1
% regexp {^((0x)?[0-9a-fA-F]+|[0-9]+)$} 0xabcd
1
% regexp {^((0x)?[0-9a-fA-F]+|[0-9]+)$} 12345
1
% regexp {^((0x)?[0-9a-fA-F]+|[0-9]+)$} 123j
0
```

如果 regexp 命令后面有参数 matchVar 和 subMatchVar, 则所有的参数被当作变量名, 如果变量不存在, 就会被生成。regexp 把匹配整个正则表达式的子字符串赋给第一个变量, 匹配正则表达式的最左边的子表达式的子字符串赋给第二个变量, 依次类推, 如例 10-57 所示。

例 10-57 regexp 命令变量

```
% regexp { ([0-9]+)*([a-z]+)} " there is 100 apples" total num word
1
% puts " $total , $num, $word"
100 apples ,100,apples
```

regexp 可以设置一些开关 (switchs), 来控制匹配结果:

1) -nocase 匹配时不考虑大小写。

2) -indices 改变各个变量的值, 这是各个变量的值变成了对应的匹配子串在整个字符串中所处位置的索引, 如例 10-58 所示。

例 10-58 regexp 命令参数

```
% regexp -indices { ([0-9]+) *([a-z]+) } " there is 100 apples" total num word
1
% puts " $total , $num, $word"
9 20 ,10 12,15 20
```

正好子串 “100 apples” 的序号是 9 ~ 20, “100” 的序号是 10 ~ 12, “apples” 的序号是 15 ~ 20。

“--” 表示这后面再没有开关 (switch) 了, 即使后面有以 “-” 开头的参数也被当作正则表达式的一部分。

4. regsub 命令

语法:

```
regsub ?switchs? exp string subSpec varname
```

regsub 命令的第一个参数是一个正则表达式, 第二个参数是一个输入字符串, 这一点和 regexp 命令完全一样, 也是当匹配时返回 1, 否则返回 0。不过 regsub 用第三个参数的值来替换字符串 string 中和正则表达式匹配的部分, 第四个参数被认为是一个变量, 替换后的字符串存入这个变量中, 如例 10-59 所示。

例 10-59 regsub 命令

```
% regsub there "They live there lives " their x
1
% puts $x
They live their lives
```

例 10-59 中 there 被用 their 替换了。

regsub 命令也有几个开关:

1) -nocase 意义同 regexp 命令中。

2) -all 没有这个开关时, regsub 只替换第一个匹配, 有了这个开关, regsub 将把所有匹配的地方全部替换。

5. string 命令

语法:

```
string option arg ?arg...?
```

string 命令具有强大的操作字符串的功能, 其中的 option 选项多达 20 个。下面介绍其中常用的选项。

(1) string compare ?-nocase? ?-length int? string1 string2

把字符串 string1 和 string2 进行比较, 返回值为 -1、0 或 1, 分别对应 string1 小于、等于或大于 string2。如果有 -length 参数, 那么只比较前 int 个字符, 如果 int 为负数, 那么这个参数被忽略。如果有 -nocase 参数, 那么比较时不区分大小写。

(2) string equal ?-nocase? ?-length int? string1 string2

把字符串 string1 和 string2 进行比较, 如果两者相同, 返回值为 1, 否则返回 0。

(3) string first string1 string2 ?startindex?

在 string2 中从头查找与 string1 匹配的字符序列, 如果找到, 那么就返回匹配的的第一个字母所在的位置; 如果没有找到, 那么返回 -1。如果给出了 startindex 变量, 那么将从 startindex 处开始查找, 如例 10-60 所示。

例 10-60 string 命令用法 1

```
% string first ab defabc
3
% string first ab defabc 4
-1
```

(4) string last string1 string2 ?startindex?

last 参数与 first 参数的唯一区别是从后往前查找。

(5) string index string1 charIndex

返回 string 中第 charIndex 个字符。charIndex 可以是下面的值:

□ 整数 n : 字符串中第 n 个字符。

□ end: 最后一个字符。

□ end- 整数 n : 倒数第 n 个字符。string index "abcd" end-1 返回字符 'c'。

如果 charIndex 小于 0, 或者大于字符串 string 的长度, 那么返回空。

如例 10-61 所示。

例 10-61 string 命令用法 2

```
% string index abcdef 2
c
% string index abcdef end-2
d
```

(6) string length string

返回字符串 string 的长度。

(7) string match ?-nocase? pattern string

如果 pattern 匹配 string, 那么返回 1, 否则返回 0。如果有 -nocase 参数, 那么就不区分

大小写。

在 pattern 中可以使用通配符如下：

- * 匹配 string 中的任意长的任意字符串，包括空字符串。
- ? 匹配 string 中任意单个字符。
- [chars] 匹配字符集合 chars 中给出的任意字符，其中可以使用 A-Z 这种形式。
- \x 匹配单个字符 x，使用 \ 是为了让 x 可以为字符 *、-、[、]，如例 10-62 所示。

例 10-62 string 命令用法 3

```
% string match * abcdef
1
% string match a* abcdef
1
string match a?cdef abcdef
1
% string match {a[b-f]cdef} abcdef    * 注意一定要用 '{'，否则 Tcl 解释器会把 b-f
                                         * 当作命令名，从而导致错误
1
% string match {a[b-f]cdef} accdef
1
```

(8) string range string1 first last

返回字符串 string1 中从第 first 个到第 last 个字符的子字符串。如果 first<0，那么 first 被看作 0；如果 last 大于或等于字符串的长度，那么 last 被看作 end；如果 first 比 last 大，那么返回空。

(9) string repeat string1 count

返回值为重复了 string1 字符串 count 次的新字符串，如例 10-63 所示。

例 10-63 string 命令用法 4

```
% string repeat "abc" 2
abccabc
```

(10) string replace string1 first last ?newstring?

返回值为从字符串 string1 中删除了第 first 到第 last 个字符的新字符串，如果给出了 newstring 变量，那么就用 newstring 替换从第 first 到第 last 个字符。如果 first<0，那么 first 被看作 0；如果 last 大于或等于字符串的长度，那么 last 被看作 end；如果 first 比 last 大或者大于字符串 string1 的长度或者 last 小于 0，那么原封不动地返回 string1。

(11) string tolower string1 ?first? ?last?

返回值为把字符串 string1 转换成小写字母后的字符串，如果给出了 first 和 last 变量，就只转换 first 和 last 之间的字符。

(12) string toupper string1 ?first? ?last?

与 tolower 参数的唯一区别是转换成大写字母。

(13) string trim string1 ?chars?

返回值是：从 string1 字符串的首尾删除掉了字符集合 chars 中的字符后的字符串。如果没有给出 chars，那么将删除掉 spaces、tabs、newlines、carriage returns 这些字符，如例 10-64 所示。

例 10-64 string 命令用法 5

```
% string trim "abcde" {a d e}
bc
% string trim " def > "
def
```

(14) string trimleft string1 ?chars?

与参数 trim 的唯一不同是只删除左边的字符。

(15) string trimright string1 ?chars?

与参数 trim 的唯一不同是只删除右边的字符。

10.10 文件访问

Tcl 提供了丰富的文件操作的命令。通过这些命令，可以对文件名进行操作（查找匹配某一模式的文件）、以顺序或随机方式读写文件、检索系统保留的文件信息（如最后访问时间）。

10.10.1 文件名

Tcl 中的文件名和我们熟悉的 Windows 表示文件的方法有一些区别：在表示文件的目录结构时它使用 '/', 而不是 '\', 这和 Tcl 最初是在 UNIX 下实现有关。比如，C 盘 Tcl 目录下的文件 sample.Tcl 在 Tcl 中这样表示：C:/Tcl/sample.Tcl。

10.10.2 基本文件输入 / 输出命令

Tcl 支持缓存机制的文件 I/O 操作。最简单的文件操作是 gets 和 puts，但当有大量数据需要读取时，read 命令更有效，可以通过 read 命令将整个文件数据都读出来。下面介绍上述过程中用到的几个基本的文件输入输出命令。

1. open 命令

```
open name ?access?
```

open 命令以 access 方式打开文件 name，返回供其他命令（gets、close 等）使用的文件标识。如果 name 的第一个字符是 "|”，管道命令被触发，而不是打开文件。

文件的打开方式和我们熟悉的 C 语言类似，有以下方式：

□ r：只读方式打开文件。文件必须已经存在。这是默认方式。

□ **r+**: 读写方式打开文件, 文件必须已经存在。

□ **w**: 只写方式打开文件, 如果文件存在则清空文件内容, 否则创建一个新的空文件。

□ **w+**: 读写方式打开文件, 如文件存在则清空文件内容, 否则创建新的空文件。

□ **a**: 只写方式打开文件, 文件必须存在, 并把文件指针指向文件尾。

□ **a+**: 读写方式打开文件, 并把文件指针指向文件尾。如文件不存在, 创建新的空文件。

open: 命令返回一个字符串用于标识打开的文件。当调用别的命令 (如: **gets**、**puts**、**close**) 对打开的文件进行操作时, 就可以使用这个文件标识符。Tcl 有 3 个特定的文件标识: **stdin**、**stdout** 和 **stderr**, 分别对应标准输入、标准输出和错误通道, 任何时候你都可以使用这 3 个文件标识。

2. gets 命令

```
gets    fileId    ?varName?
```

读 **fileId** 标识的文件的下一行, 忽略换行符。如果命令中有 **varName** 就把该行赋给它, 并返回该行的字符数 (文件尾返回 -1); 如果没有 **varName** 参数, 返回文件的下一行作为命令结果 (如果到了文件尾, 就返回空字符串)。

3. read 命令

read 是和 **gets** 类似的命令, 不过 **read** 不是以行为单位读的, 它有两种形式:

(1) **read ?-nonewline? fileId**

读并返回 **fileId** 标识的文件中所有剩下的字节。如果没有 **nonewline** 开关, 则在换行符处停止。

(2) **read fileId numBytes**

在 **fileId** 标识的文件中读, 并返回下一个 **numbytes** 字节。

4. puts 命令

```
puts    ?-nonewline?    ?fileId? string
```

puts 命令把 **string** 写到 **fileId** 中, 如果没有 **nonewline** 开关的话, 添加换行符。**fileId** 默认是 **stdout**。命令返回值为一个空字符串。

puts 命令使用 C 的标准 I/O 库的缓冲区方案, 这就意味着使用 **puts** 产生的信息不会立即出现在目标文件中。

5. flush 命令

如果你想使数据立即出现在文件中, 那就调用 **flush** 命令。

```
flush    fileId
```

把缓冲区内容写到 **fileId** 标识的文件中, 命令返回值为空字符串。

flush 命令强迫把缓冲区中的数据写到文件中。**flush** 直到数据被写完才返回。当文件关闭时缓冲区数据会自动 **flush**。

6. close 命令

```
close ?fileId?
```

关闭标识为 `fileId` 的文件，命令返回值为一个空字符串。

这里特别要说明的一点是，Tcl 中对串口、管道、socket 等的操作和对文件的操作类似，以上对文件的操作命令同样适用于它们。

这个名为 `tgrep` 的过程，可以说明 Tcl 文件 I/O 的基本特点，如例 10-65 所示。

例 10-65 `tgrep` 过程程序

```
proc tgrep { pattern filename } {  
    set f [open $filename r]  
    while { [gets $f line] } {  
        if {[regexp $pattern $line]} {  
            puts stdout $line  
        }  
    }  
    close $f  
}
```

例 10-65 的过程非常像 UNIX 的 `grep` 命令，可以用两个参数调用它，一个是模式，另一个是文件名，`tgrep` 将打印出文件中所有匹配该模式的行。

第 11 章

Tcl 脚本编程应用实例 (PT 篇)

本章将基于 Synopsys 公司的 PrimeTime 工具 (简称 PT) 来讲解静态时序分析中的 Tcl 脚本应用实例。要想编写高质量的 Tcl 脚本, 需要熟练掌握和了解 PT 工具的所有命令和用法。但是本章的学习目的是使读者快速掌握如何编写基本的 Tcl 脚本, 因此本章只对实例代码中涉及的部分 PT 命令进行讲解, 关于 PT 的所有命令的详细使用说明, 读者可以参阅 PT 的用户使用手册。

本章将由浅入深地通过 8 个 Tcl 脚本实例来讲解如何编写实用的 Tcl 脚本, 从而使读者具备通过编写 Tcl 脚本来达到提高工作效率和质量的基本能力。

在本章的例子中, 注释说明文字用 * 与代码隔开。

11.1 get_failing_paths_high_slew

该程序用于获得时序违反路径中违反路径转换时间的阈值节点的信息, 其具体的代码含义将在下面讲解。

```
proc get_failing_paths_high_slew { Nworst_num Path_type Tran_threshold }{
*****
    定义名为 get_failing_paths_high_slew 的过程, 其中包含 3 个输入参数变量: 输入变量 Nworst_
    numb 为违反路径的数量值; 输入变量 Path_type 为实现分析的违反路径的类型, 比如建立时间或保持时间; 输
    入变量 Tran_threshold 为违反路径转换时间的阈值。
*****
    set Nworst_num $Nworst_num
*****
    将过程的输入变量 Nworst_num 的值传递给内部的局部变量 Nworst_num。
*****
    set Path_type $Path_type
*****
    将过程的输入变量 Path_type 的值传递给内部的局部变量 Path_type。
*****
    set Tran_threshold $Tran_threshold
*****
    将过程的输入变量 Tran_threshold 的值传递给内部的局部变量 Tran_threshold。
*****
}
```

```
foreach_in_collection path [get_timing_paths -delay_type $Path_type -nworst $Nworst_num] {
```

通过 `get_timing_paths` 获得时序分析结果，再通过 `foreach_in_collection` 命令循环地获取分析结果，并赋值到变量 `path` 中，其中变量 `path` 的值为存储了一条时序路径数据的指针地址。为了区分不同的循环程序，该循环程序人为定义为循环程序 0。

`foreach_in_collection` 命令是 PT 工具的命令，其功能类似于 Tcl 中的循环命令 `foreach`，用于循环地逐个提取链表的单元，但是不同的是 `foreach` 命令接收的输入参数值是一个列表，而 `foreach_in_collection` 命令接收的输入参数值是列表的指针地址。由于 PT 工具很多命令返回的变量值是指针地址，因此使用 PT 自带的循环命令 `foreach_in_collection` 不需要对返回指针地址进行提取并转换成 Tcl 的输入变量格式的过程，这样可以提高代码的执行效率。

`get_timing_paths` 命令是 PT 工具的命令，用于获得时序分析结果，其返回值为指针地址。其中参数 `-delay_type` 定义时序分析的类型是建立时间或保持时间；参数 `-nworst` 定义时序分析结果中包含时序违反最严重的路径数量。

```
foreach_in_collection itr $path {
```

通过 `foreach_in_collection` 命令循环地获取变量 `path` 的值，并将变量 `path` 的值赋值到变量 `itr` 中。为了区分不同的循环程序，该循环程序人为定义为循环程序 1。

```
set cnt 0
```

设置变量 `cnt` 的值为 0。

```
set Slack [get_attribute $itr slack]
```

通过 `get_attribute` 命令获得变量 `itr` 的时序违反值，并赋值给变量 `Slack`。

`get_attribute` 命令是 PT 工具中最常用的命令，它用于得到输入变量值的属性信息，但返回的结果是地址指针。其属性信息在命令的参数中进行定义，这里属性信息为时序违反值 `slack`。

```
set StartPoint [get_object_name [get_attribute $itr startpoint] ]
```

`get_attribute` 命令中的参数值为 `startpoint`，说明获得时序路径中起点单元的信息。

`get_object_name` 命令是 PT 工具的命令，通过该命令获得变量 `itr` 的时序路径起点单元的名称，并赋值给变量 `StartPoint`。

```
set EndPoint [get_object_name [get_attribute $itr endpoint] ]
```

`get_attribute` 命令中的参数值为 `endpoint`，说明获得时序路径中终点单元的信息。通过 `get_object_name` 命令获得变量 `itr` 的时序路径终点单元的名称，并赋值给变量 `EndPoint`。

```
if { $Slack < 0.0 }{
```

如果时序违反值小于 0，则 `if` 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义为分支程序 0。

```
if { $cnt == 0 }{
```

如果变量 `cnt` 等于 0，则 `if` 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义为分支程序 1。

```

*****
echo " Timing path fails:  \t Start point: $StartPoint \t End Point $EndPoint \t
Slack: $Slack"
*****
    输出字符串信息, 同时把对应的变量值显示到字符串信息中。该字符串信息主要是把该条时序违反路径的起
    点单元、终点单元和时序违反值通过显示终端传递给用户。
*****

    set $cnt [ expr $cnt + 1]
*****
    对变量 cnt 进行加 1 操作。
*****
}
*****
    结束分支程序 1。
*****

    foreach_in_collection point [get_attribute $path points] {
*****
        get_attribute 命令中的参数值为 points, 说明获得时序路径的节点单元的所有信息, 再通过
        foreach_in_collection 命令循环地获取节点单元, 并赋值到变量 point 中。为了区分不同的循环程序, 该循
        环程序人为定义为循环程序 2。
*****

        set Object [get_attribute $point object]
*****
        get_attribute 命令中的参数值为 object, 说明获得节点单元的物体指针信息, 再赋值给变量 Object。
*****

        set Obj_name [get_object_name [get_attribute $point object] ]
*****
        基于节点单元的物体指针信息再通过 get_object_name 命令获得节点单元的名称, 并赋值给变量 Obj_
        name。
*****

        set Obj_r_f [get_attribute $point rise_fall]
*****
        get_attribute 命令中的参数值为 rise_fall, 说明获得节点单元的信号转换状态信息, 再赋值给变量
        Obj_r_f。
*****

        set Slew_att [format "actual_%s_transition_%s" $Obj_r_f $Path_type]
*****
        将格式化后的字符串信息赋值给 Slew_att。
*****

        set Slew_val [ get_attribute $Object $Slew_att]
*****
        get_attribute 命令中的参数值为变量 Slew_att 的值, 说明获得节点单元的信号转换信息, 并赋值给变
        量 Slew_val。
*****

        set Cell [ get_cells -quiet -of_objects $Object ]
*****
        get_cells 命令是 PT 工具的命令, 通过该命令可以获得节点单元的单元信息, 但返回的结果是地址指针。
        这里将获得的单元信息赋值给变量 Cell。
*****

        if { [get_attribute -quiet $Object is_port] == "false" }{

```

如果变量 Object 的属性不是端口，则 if 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义为分支程序 2。

```
set Lib_cell [ get_lib_cells -of_objects $Cell ]
```

get_lib_cells 命令是 PT 工具的命令，通过该命令可以获得库单元信息，但返回的结果是地址指针。这里获得变量 Cell 的库单元信息并赋值给变量 Lib_cell。

```
set Ref_name [ get_attribute $Lib_cell base_name ]
```

get_attribute 命令中的参数值为 base_name，说明获得节点单元的实例名信息，再赋值给变量 Ref_name。

```
} else {
```

结束分支程序 2，如果分支程序 2 由于判断条件没有满足而没有执行，则执行分支程序 3。

```
set Ref_name $Object
```

把变量 Object 值赋值给变量 Ref_name。

```
}
```

结束分支程序 3。

```
if { $Slew_val > $Tran_threshold }{
```

如果变量 Slew_val 的值大于变量 Tran_threshold，则执行分支 4。

```
echo " has a $Obj_r_f slew = $Slew_val on pin $Obj_name \t Reference = $Ref_name"
```

输出字符串信息，同时把对应的变量值显示到字符串信息中。该字符串信息主要是把该条时序违反路径的节点的信息，包括库单元信息、信号转换信息和实例信息通过显示终端传递给用户。

```
}
```

结束分支程序 4。

```
}
```

结束循环程序 2。

```
} else {
```

结束分支程序 0，如果分支程序 0 由于判断条件没有满足而没有执行，则执行分支程序 5。

```
continue
```

由于判断条件没有满足执行分支程序 0 的条件，说明该时序路径的时序违反值大于 0，因此跳过本次循环程序 2。

```

*****
}
*****
结束分支程序 5。
*****
}
*****
结束循环程序 1。
*****
}
*****
结束循环程序 0。
*****
unset Slack Obj_name Obj_r_f Slew_att Slew_val Nworst_numb Path_type Tran-
threshold
*****
取消变量定义。
*****
}
*****
过程 get_failing_paths_high_slew 结束。
*****

```

输入命令 `get_failing_paths_high_slew 2 min 0` 后, 其结果如图 11-1 所示。

```

pt_shell> get_failing_paths_high_slew 2 min 0
Timing path fails:      Start point: arst_l      End Point fpu_add/fpu_add_ctl/dffrl_add_ctl/q_reg[0]/RN      Slack: -0.038200
Timing path fails:      Start point: arst_l      End Point fpu_div/fpu_div_ctl/dffrl_div_ctl/q_reg[0]/RN      Slack: -0.038200
Timing path fails:      Start point: cluster_cken      End Point cluster_header/I0/sync_cluster_master/q_r_reg/D      Slack: -0.059595
Timing path fails:      Start point: global_shift_enable      End Point cluster_header/I0/rst_repeater/repeater/I0/q_reg/D      Slack: -0.030500
has a fall slew = 0.093195 on pin cluster_header/I0/rst_repeater/repeater/I0/U4/Y      Reference = NOR2BX1
has a fall slew = 0.093195 on pin cluster_header/I0/rst_repeater/repeater/I0/q_reg/D      Reference = DFFSRHQX1

```

图 11-1 `get_failing_paths_high_slew 2 min 0` 命令结果

11.2 get_interclock_skew

该程序用于获得交叉时钟之间的时钟偏斜信息, 其具体的代码含义将在下面详细讲解。

```

proc get_interclock_skew {args}{
*****
    定义名为 get_interclock_skew 的过程, 其中包含 1 个输入参数变量 args。
*****
    set results(-delay_type) {max}
*****
    定义名为 results 的数组, 其数组元素的名字为 -delay_type, 其值为 max。
*****
    parse_proc_arguments -args $args results
*****

```

调用 PT 内嵌的名为 `parse_proc_arguments` 的过程, 该过程的作用是提取主过程程序的输入变量, 并将输入变量的值赋给过程程序中对应的局部变量 `results`, 同时用户还可以通过输入参数 `-help` 来查询主过程程序的使用说明。


```

*****
    if {[![info exists results(-from)] || $results(-from) eq {}} && {[info exists
results(-to)] || $results(-to) eq {}} ){
*****
    如果输入变量值缺失或者不对，则 if 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人
为定义为分支程序 0。
*****
    echo "Error: at least one of -from, -to must be specified."
*****
    将输出的字符串信息通过显示终端传递给用户，该字符串信息说明输入数据不正确。
*****
    return {}
*****
    返回一个空列表。
*****
}
*****
结束分支程序 0。
*****
regexp {^(...)} $results(-delay_type) mm
*****
对数组元素 results(-delay_type) 的值进行字符串匹配，并把匹配的值赋值给变量 mm。
*****
set command "get_timing_paths"
*****
定义变量 command 的值。
*****
if {[info exists results(-from)]}{append command " -from $results(-from)"}
*****
如果数组元素 results(-from) 存在数值，则把其数值追加到变量 command 的字符串尾。
*****
if {[info exists results(-to)]}{append command " -to $results(-to)"}
*****
如果数组元素 results(-to) 存在数值，则把其数值追加到变量 command 的字符串尾。
*****
eval set paths \[$command\]
*****
定义变量 paths 的值，其值即为一个完整的 get_timing_paths 命令的结果。
*****
if {[sizeof_collection $paths ] == 0 }{
*****
sizeof_collection 命令是 PT 工具的命令，通过该命令可以得到输入变量元素的数量值。
这里通过命令 sizeof_collection 来得到执行命令 get_timing_paths 后的返回结果个数。
如果返回结果个数为 0，则 if 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义
为分支程序 1。
*****
echo "No constrained paths"
*****
输出字符串信息通过显示终端传递给用户，该字符串信息主要是说明没有时序约束正确的路径。
*****
return 1

```

返回一个数值 1。

}

结束分支程序 1。

```
set startpoint_clock_latency [get_attribute $paths startpoint_clock_latency]
```

get_attribute 命令中的参数值为 startpoint_clock_latency, 说明获得时序路径的起点单元的时钟延迟信息, 再赋值给变量 startpoint_clock_latency。

```
set endpoint_clock_latency [get_attribute $paths endpoint_clock_latency]
```

get_attribute 命令中的参数值为 endpoint_clock_latency, 说明获得时序路径的终点单元的时钟延迟信息, 再赋值给变量 endpoint_clock_latency。

```
set crpr [get_attribute $paths common_path_pessimism]
```

get_attribute 命令中的参数值为 common_path_pessimism, 说明获得时序路径的时钟路径悲观移除信息, 再赋值给变量 crpr。

```
if { $mm == "max" }{
```

如果变量 mm 的值为 max, 则 if 控制流命令执行该分支程序。为了区分不同的分支程序, 该分支程序人为定义为分支程序 2。

```
set interclock_skew [expr $startpoint_clock_latency - $endpoint_clock_latency - $crpr]
```

通过命令 expr 和对应的数学表达式计算出时钟偏斜值并赋给变量 interclock_skew。

} else {

结束分支程序 2, 如果分支程序 2 由于判断条件没有满足而没有执行, 则执行分支程序 3。

```
set interclock_skew [expr $endpoint_clock_latency - $startpoint_clock_latency + $crpr]
```

通过命令 expr 和对应的数学表达式计算出时钟偏斜值并赋给变量 interclock_skew。

}

结束分支程序 3。

}

过程 get_interclock_skew 结束。

```
define_proc_attributes get_interclock_skew \
```

```

-info "Reports interclock skew for max path"\
-define_args {\
  {-from "From pins or ports" from_list string }
  {-to "To pins or ports" to_list string }
  {-delay_type "Type of delay Default= max" type one_of_string {optional value_
help {values {min max }}}}}
}

```

PT 内嵌的名为 `define_proc_attributes` 的过程，该过程的作用是定义对应过程的帮助说明，对过程的输入变量进行说明，这样用户可以通过输入参数 `-help` 来查询主过程程序的使用说明，该过程需要与 PT 内嵌的名为 `parse_proc_arguments` 的过程一起使用，否则程序会出错。这里通过 `define_proc_attributes` 的过程对输入变量 `-from`、`-to` 和 `-delay_type` 进行了说明定义，同时通过参数 `-info` 可以定义过程的应用说明。

输入 `get_interclock_skew -help`，其结果如图 11-2 所示。

```

pt_shell> get_interclock_skew -help
Usage:
get_interclock_skew # Reports interclock skew for max path
  -from from_list      (From pins or ports)
  -to to_list          (To pins or ports)
  [-delay_type type]   (Type of delay Default= max:
                       Values: min, max)

```

图 11-2 `get_interclock_skew` 参数信息

输入命令 `get_interclock_skew -to cluster_header/I0/U2 -from cluster_header/I0/sync_cluster_slave/so_l_reg -delay_type min`，其结果如图 11-3 所示。

```

pt_shell> get_interclock_skew -to cluster_header/I0/U2 -from cluster_header/I0/sync_cluster_slave/so_l_reg
-delay_type min
0.1
pt_shell> █

```

图 11-3 `get_interclock_skew -to cluster_header/I0/U2 -from cluster_header/I0/sync_cluster_slave/so_l_reg -delay_type min` 命令结果

11.3 report_unclocked

该程序用于获得没有时钟输入的时间单元信息，其具体的代码含义将在下面解释。

```

proc report_unclocked {args}{
*****
  定义名为 report_unclocked 的过程，其中包含 1 个输入参数变量 args。
*****
  set procargs(-verbose) false
*****
  定义名为 procargs 的数组，其数组元素的名字为 -verbose，其值为 false。
*****
  parse_proc_arguments -args $args procargs
*****
  调用 PT 内嵌的名为 parse_proc_arguments 的过程，提取主过程程序的输入变量并把输入变量的值赋给

```

过程程序中对应的局部变量 `procargs`。

```
*****
set verbose $procargs(-verbose)
*****
```

把数组元素 `procargs(-verbose)` 的值赋给变量 `verbose`。

```
*****
set all_clock_pins [all_registers -clock_pins]
*****
```

`all_registers` 命令是 PT 工具的命令, 通过命令 `all_registers` 获得所有定义为时钟的 Pin 脚, 但返回的结果是地址指针。

这里把获得的所有时钟 Pin 脚结果赋给变量 `all_clock_pins`。

```
*****
if {$::synopsys_program_name == "pt_shell"}{
*****
```

如果全局变量 `::synopsys_program_name` 的值等于 `pt_shell`, 则 `if` 控制流命令执行该分支程序。为了区分不同的分支程序, 该分支程序人为定义为分支程序 0。

```
*****
set unlocked0 [filter_collection $all_clock_pins "defined(clocks)"]
*****
```

`filter_collection` 命令是 PT 工具的命令, 通过该命令获得所有实际连接时钟信号的 Pin 脚, 但返回的结果是地址指针。

这里把获得的所有实际连接时钟信号的 Pin 脚赋给变量 `unlocked0`。

```
*****
set unlocked [remove_from_collection $all_clock_pins $unlocked0]
*****
```

`remove_from_collection` 命令是 PT 工具的命令, 通过该命令移除指定列表中的指定元素单元, 但返回的结果是地址指针。

这里把 `unlocked0` 的元素从变量 `all_clock_pins` 中移除, 并把移除后剩下的结果赋给变量 `unlocked`。

```
*****
set constant {Logic([01])/output}
*****
```

定义变量 `constant` 的值。

```
*****
} else {
*****
```

结束分支程序 0, 如果分支程序 0 由于判断条件没有满足而没有执行, 则执行分支程序 1。

```
*****
set unlocked [filter_collection $all_clock_pins is_on_clock_network==false]
*****
```

将通过 `filter_collection` 命令获得的所有没有实际连接时钟信号的 Pin 脚赋给变量 `unlocked`。

```
*****
set constant {\*\*logic_{[01]}\*\*}
*****
```

定义变量 `constant` 的值。

```
*****
}
*****
```

结束分支程序 1。

```
*****
array unset uc_roots
*****
```

取消数组变量 uc_roots 的定义。

```
*****
set uc_roots(LOGIC_0) {}
*****
    定义名为 uc_roots 的元素 LOGIC_0 的值为空。
*****
set uc_roots(LOGIC_1) {}
*****
    定义名为 uc_roots 的元素 LOGIC_1 的值为空。
*****
set c_root ""
*****
    定义变量 c_root 为空。
*****
foreach_in_collection uc $unclocked {
*****
    通过 foreach_in_collection 命令循环地获取变量 unclocked 的值，并将变量 unclocked 的值赋值
    到变量 uc 中。为了区分不同的循环程序，该循环程序人为定义为循环程序 0。
*****
    redirect -variable redir { set c_root [get_object_name [all_fanin -flat
    -startpoints_only -to $uc -trace_arcs all]] }
*****
    redirect 命令是 PT 工具的命令，通过该命令可以把其他命令的输出结果重新指定到新的输出文件中。
    all_fanin 命令是 PT 工具的命令，通过该命令可以得到指定输入 Pin 脚的驱动端 Pin 脚。
    这里通过 redirect 命令把 all_fanin 命令的结果重新赋给变量 redir。
*****
    if {$::synopsys_program_name == "icc_shell"&& $c_root==""}{
*****
        如果全局变量 ::synopsys_program_name 的值等于 icc_shell，同时变量 c_root 为空，则 if 控制流
        命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义为分支程序 2。
*****
        set c_root $redir
*****
        定义变量 c_root。
*****
    }
*****
    结束分支程序 2。
*****
    if {[regexp $constant $c_root m s]}{
*****
        如果字符串匹配成功，则 if 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义
        为分支程序 3。
*****
        set c_root "LOGIC_$s"
*****
        定义变量 c_root。
*****
    }
*****
    结束分支程序 3。
*****
```

```

*****
lappend uc_roots($c_root) [get_object_name $uc]
*****
把 get_object_name 的返回结果追加到数组元素 uc_roots($c_root) 的列表中。
*****
}
*****
结束循环程序 0。
*****
echo "List of clock drivers and number of unlocked clock pins in their fanout:\n"
*****
将输出的字符串信息通过显示终端传递给用户, 该字符串信息提示没有实际连接时钟信号的 Pin 脚信息。
*****
set all_roots [concat [lminus [lsort [array names uc_roots]] {LOGIC_0 LOGIC_1}]
{LOGIC_0 LOGIC_1}]
*****
lminus 命令是 PT 工具的命令, 通过该命令在指定的列表中移除指定的元素。
这里移除列表 uc_roots 中的指定元素, 赋值给变量 all_roots。
*****
foreach c_root $all_roots {
*****
通过 foreach 命令循环地获取变量 all_roots 的值, 并将变量 all_roots 的值赋值到变量 c_root 中。
为了区分不同的循环程序, 该循环程序人为定义为循环程序 1。
*****
echo [format "%-50s %d" $c_root [llength $uc_roots($c_root)]]
*****
将输出的字符串信息通过显示终端传递给用户, 该字符串信息是整理出没有实际连接时钟信号的 Pin 脚实际
信息。
*****
if {$verbose}{
*****
如果存在变量 $verbose, 则 if 控制流命令执行该分支程序。为了区分不同的分支程序, 该分支程序人为定
义为分支程序 4。
*****
foreach pin $uc_roots($c_root) {
*****
通过 foreach 命令循环地获取数组元素 $uc_roots($c_root) 的值, 并赋值到变量 pin 中。为了区分不
同的循环程序, 该循环程序人为定义为循环程序 2。
*****
echo " $pin"
*****
输出字符串变量 pin 的信息, 并通过显示终端传递给用户。
*****
}
*****
结束循环程序 2。
*****
}
*****
结束分支程序 4。

```

```

*****
}
*****
结束循环程序 1。
*****
}
*****
过程 report_unlocked 结束。
*****
define_proc_attributes report_unlocked \
-info "Report unlocked registers"\
-define_args {
{-verbose "Be verbose." "" boolean optional}
}
*****
PT 内嵌的名为 define_proc_attributes 的过程，通过 define_proc_attributes 的过程对输入变
量 -verbose 进行说明定义。
*****

```

输入 report_unlocked -help，其结果如图 11-4 所示。

```

pt_shell> report_unlocked -help
Usage:
report_unlocked      # Report unlocked registers
[-verbose]           (Be verbose.)

```

图 11-4 report_unlocked 参数信息

输入 report_unlocked，其结果如图 11-5 所示。

```

pt_shell> report_unlocked
List of clock drivers and number of unlocked clock pins in their fanout:
arst_l {i_fpu_inq_sram/wrptr_dl_reg[2]/CK} {i_fpu_inq_sram/wrptr_dl_reg[1]/CK} {i_fpu_inq_sram/wrptr_dl_reg[3]/CK} {i_fpu_inq_sram/wrptr_dl_reg[0]/CK} i_fpu_
inq_sram/wr_en_dl_reg/CK global_shift_enable ctu_tst_pre_grat_l 155
i_fpu_inq_sram/ren_dl_reg/CK 155

```

图 11-5 report_unlocked 命令结果

11.4 get_buffers

该程序用于获得缓冲单元的信息，其具体的代码含义如下。

```

proc get_buffers {args}{
*****
定义名为 get_buffers 的过程，其中包含 1 个输入参数变量 args。
*****
parse_proc_arguments -args ${args} options
*****
调用 PT 内嵌的名为 parse_proc_arguments 的过程，提取主过程程序的输入变量，并把输入变量的值赋
值给过程程序中对应的局部变量 options。
*****
if {[info exists options(-library)]}{

```

```

*****
    如果数组元素 options(-library) 存在数值, 则 if 控制流命令执行该分支程序。为了区分不同的分支程
    序, 该分支程序人为定义为分支程序 0。
*****
    set lib_id $options(-library)
*****
    定义变量 lib_id 的值为数组变量 options(-library) 的值。
*****
} else {
*****
    结束分支程序 0, 如果分支程序 0 由于判断条件没有满足而没有执行, 则执行分支程序 1。
*****
    set lib_id *
*****
    定义变量 lib_id 的值为 *。
*****
}
*****
    结束分支程序 1。
*****
    if {[info exists options(-pattern)]}{
*****
        如果数组元素 options(-pattern) 存在数值, 则 if 控制流命令执行该分支程序。为了区分不同的分支程
        序, 该分支程序人为定义为分支程序 2。
*****
        set pattern $options(-pattern)
*****
        定义变量 pattern 的值为数组变量 options(-pattern) 的值。
*****
    } else {
*****
        结束分支程序 2, 如果分支程序 2 由于判断条件没有满足而没有执行, 则执行分支程序 3。
*****
        set pattern *
*****
        定义变量 pattern 的值为 *。
*****
    }
*****
    结束分支程序 3。
*****
    set gen_inv [info exists options(-inverter)]
*****
    定义变量 gen_inv 的值, 值为 1 表示数组元素 options(-inverter) 存在数值, 否则为 0。
*****
    set use_arc [info exists options(-use_arc_info)]
*****
    定义变量 use_arc 的值, 值为 1 表示数组元素 options(-use_arc_info) 存在数值, 否则为 0。
*****

```



```

set ver [info exists options(-verbose)]
*****
    定义变量 ver 的值，值为 1 表示数组元素 options(-verbose) 存在数值，否则为 0。
*****
    if ${gen_inv} {
*****
        如果变量 gen_inv 为 1，则 if 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义
        为分支程序 4。
*****
        set req_arc negative_unate
*****
        定义变量 req_arc 的值为 negative_unate。
*****
        set typ inverter
*****
        定义变量 typ 的值为 inverter。
*****
    } else {
*****
        结束分支程序 4，如果分支程序 2 由于判断条件没有满足而没有执行，则执行分支程序 5。
*****
        set req_arc positive_unate
*****
        定义变量 req_arc 的值为 positive_unate。
*****
        set typ buffer
*****
        定义变量 typ 的值为 buffer。
*****
    }
*****
    结束分支程序 5。
*****
    set cell_coll ""
*****
    定义变量 cell_coll 的值为一个空列表。
*****
    set libs [get_libs ${lib_id}]
*****
    get_libs 命令是 PT 工具的命令，通过该命令可以得到指定工艺库的信息，但返回的结果是地址指针。
    这里把 get_libs 命令的返回结果赋值给变量 libs。
*****
    foreach_in_collection lib ${libs} {
*****
        通过 foreach_in_collection 命令循环地获取变量 libs 的值，并赋值到变量 lib 中。为了区分不同的
        循环程序，该循环程序人为定义为循环程序 0。
*****
        set lcells [filter_collection [get_lib_cells -quiet [get_attr $lib extended_
name]/*] "number_of_pins==2 && base_name=~${pattern}"]

```

```

*****
    通过 get_lib_cells 命令获得变量 lib 的库单元信息, 整理后赋值给变量 lcells。
*****
    foreach_in_collection lcell ${lcells}{
*****
    通过 foreach_in_collection 命令循环地获取变量 lcells 的值并赋值到变量 lcell 中。为了区分不同的
    循环程序, 该循环程序人为定义循环程序 1。
*****
    set opin [get_lib_pins -quiet -of_object ${lcell} -filter "pin_direction==out"]
*****
    get_lib_pins 命令是 PT 工具的命令, 通过该命令获得在指定库单元的 Pin 脚信息。
    这里获得变量 lcell 的库单元的输出生 Pin 脚信息, 并赋值给变量 opin。
*****
    set ipin [get_lib_pins -quiet -of_object ${lcell} -filter "pin_direction==in"]
*****
    通过 get_lib_pins 命令获得变量 lcell 的库单元的输入 Pin 脚信息, 并赋值给变量 opin。
*****
    if {[sizeof_coll ${opin}]==1 && [sizeof_collection ${ipin}]==1}{
*****
    如果输入 Pin 脚和输出 Pin 脚的个数都为 1, 则 if 控制流命令执行该分支程序。为了区分不同的分支程序,
    该分支程序人为定义为分支程序 6。
*****
    set opin_name [get_attribute -quiet ${opin} base_name]
*****
    get_attribute 命令中的参数值为 base_name, 说明获得变量 opin 的实例名信息, 再赋值给变量
    opin_name。
*****
    set ipin_name [get_attribute -quiet ${ipin} base_name]
*****
    通过 get_attribute 命令获得变量 ipin 的实例名信息, 再赋值给变量 ipin_name。
*****
    set cell_func [get_attribute -quiet ${opin} function]
*****
    get_attribute 命令中的参数值为 function, 说明获得变量 opin 的功能信息, 再赋值给变量 cell_
    func。
*****
    if ${gen_invv}{
*****
    如果变量 gen_inv 为 1, 则 if 控制流命令执行该分支程序。为了区分不同的分支程序, 该分支程序人为定
    义为分支程序 7。
*****
    set cond1 [string equal "(!${ipin_name})" ${cell_func}]
*****
    比较字符串变量 ipin_name 取非后和变量 cell_func 是否相等, 并把结果赋值给变量 cond1。
*****
    set cond2 [string equal "(${ipin_name})" ${cell_func}]
*****
    比较字符串变量 ipin_name 和变量 cell_func 是否相等, 并把结果赋值给变量 cond2。
*****

```

```

set cond3 [string equal "${opin_name}" "${cell_func}]
*****
比较字符串变量 opin_name 和变量 cell_func 是否相等，并把结果赋值给变量 cond3。
*****
set cond4 [string equal "!${opin_name}" "${cell_func}]
*****
比较字符串变量 opin_name 取非后和变量 cell_func 是否相等，并把结果赋值给变量 cond4。
*****
if { ${cond1} || ${cond2} || ${cond3} || ${cond4}}{
*****
如果 4 个变量 cond1、cond2、cond3 和 cond4 中有一个为 1，则 if 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义分支程序 8。
*****
set cell_coll [append_to_collection cell_coll ${lcell}]
*****
append_to_collection 命令是 PT 工具的命令，通过该命令可以将新的列表添加到对应指定的列表中，但返回的结果是地址指针。
这里通过 append_to_collection 命令将变量 lcell 的值添加到当前变量 cell_coll 中，并把新的结果再赋值给变量 cell_coll，刷新变量 cell_coll 的值。
*****
print ${ver}"Found ${typ} cell \"[get_object_name ${lcell}]\"] based on function..."
*****
调用 print 过程，将输出的字符串信息通过显示终端传递给用户。Print 过程在后面将进行详细讲解。
*****
continue
*****
跳过本次循环程序 1。
*****
}
*****
结束分支程序 8。
*****
} else {
*****
结束分支程序 7，如果分支程序 0 由于判断条件没有满足而没有执行，则执行分支程序 9。
*****
set cond1 [string equal "${ipin_name}" "${cell_func}]
*****
比较字符串变量 ipin_name 和变量 cell_func 是否相等，并把结果赋值给变量 cond1。
*****
set cond2 [string equal "(!${opin_name})" "${cell_func}]
*****
比较字符串变量 opin_name 取非后和变量 cell_func 是否相等，并把结果赋值给变量 cond2。
*****
if { ${cond1} || ${cond2}}{
*****
如果两个变量 cond1 和 cond2 中有一个为 1，则 if 控制流命令执行该分支程序。为了区分不同的分支程序，该分支程序人为定义为分支程序 10。
*****
set cell_coll [append_to_collection cell_coll ${lcell}]

```

```

*****
    通过 append_to_collection 命令将变量 lcell 的值添加到当前变量 cell_coll 中, 并把新的结果再
    赋值给变量 cell_coll, 刷新变量 cell_coll 的值。
*****
    print ${ver}"Found ${typ} cell \"[get_object_name ${lcell}]\\" based on function..."
*****
    调用过程 print 来输出字符串信息, 通过显示终端传递给用户。
*****
    continue
*****
    跳过本次循环程序 1。
*****
}
*****
    结束分支程序 10。
*****
}
*****
    结束分支程序 9。
*****
    if ${use_arc}{{
*****
        如果变量 use_arc 为 1, 则 if 控制流命令执行该分支程序。为了区分不同的分支程序, 该分支程序人为定义
        为分支程序 11。
*****
        set lib_arcs [get_lib_timing_arcs -quiet -from ${ipin} -to ${opin}]
*****
        get_lib_timing_arcs 命令是 PT 工具的命令, 通过该命令可以得到时序弧信息, 但是该命令的返回值是
        地址指针。
        这里通过 get_lib_timing_arcs 命令得到指定库单元输入到输出的时序弧信息, 并赋值给变量 lib_arcs。
*****
        foreach_in_collection larc ${lib_arcs}{
*****
            通过 foreach_in_collection 命令循环地获取变量 lib_arcs 的值, 并赋值到变量 larc 中。为了区分
            不同的循环程序, 该循环程序人为定义为循环程序 2。
*****
            set arc_sns [get_attribute -quiet ${larc} sense]
*****
            get_attribute 命令中的参数值为 sense, 说明获得变量 larc 的逻辑状态信息, 再赋值给变量 arc_sns。
*****
            if {[string match ${arc_sns} ${req_arc}]}{
*****
                如果变量 arc_sns 和变量 req_arc 的值不匹配, 则 if 控制流命令执行该分支程序。为了区分不同的分支
                程序, 该分支程序人为定义为分支程序 12。
*****
                break
*****
            } else {

```

```

*****
    结束分支程序 12，如果分支程序 0 由于判断条件没有满足而没有执行，则执行分支 13。
*****
    set cell_coll [append_to_collection cell_coll ${lcell}]
*****
    通过 append_to_collection 命令将变量 lcell 的值添加到当前变量 cell_coll 中，并把新的结果再
    赋值给变量 cell_coll，刷新变量 cell_coll 的值。
*****
    print ${ver}"Found ${typ} cell \"[get_object_name ${lcell}]" based on arc
    information..."
*****
    调用过程 print 来输出字符串信息，通过显示终端传递给用户。
*****
}
*****
    结束分支程序 13。
*****
}
*****
    结束循环程序 2。
*****
}
*****
    结束分支程序 11。
*****
}
*****
    结束分支程序 6。
*****
}
*****
    结束循环程序 1。
*****
}
*****
    结束循环程序 0。
*****
    return [append_to_collection -unique cell_coll ""]
*****
    返回变量 cell_coll 的值。
*****
}
*****
    过程 get_buffers 结束。
*****
proc print { ver msg }{
*****
    定义名为 print 的过程，其中包含两个输入参数变量：ver 和 msg。
*****
    if { $ver }{ echo "$msg"}

```

```

*****
如果变量 ver 的值为 1, 则通过显示终端输出变量 msg 的值给用户。
*****
}

*****
过程 print 结束。
*****

define_proc_attributes get_buffers \
-info "create a collection of buffer (or inverter) cells from library"\
-hide_body \
-define_arg {\
    { -library          "library from which get to buffer or inverter cells" "lib_name"
string optional }
    { -pattern          "name pattern for buffer or inverter cells" "lib_name" string
optional }
    { -inverter         "create a collection of inverter cells instead of buffers" ""
boolean optional }
    { -use_arc_info     "look at timing arc information to determine buffer/inverter" ""
boolean optional }
    { -verbose          "print detailed information" "" boolean optional }
}

*****
PT 内嵌的名为 define_proc_attributes 的过程, 通过 define_proc_attributes 的过程对输入变
量 -library、-pattern、-inverter、-use_arc_info 和 -verbose 进行说明定义。
*****

```

输入 `get_buffers -help`, 其结果如图 11-6 所示。

```

pt_shell> get_buffers -help
Usage:
get_buffers          # create a collection of buffer (or inverter) cells from library
[-library lib_name] (library from which get to buffer or inverter cells)
[-pattern lib_name] (name pattern for buffer or inverter cells)
[-inverter]         (create a collection of inverter cells instead of buffers)
[-use_arc_info]     (look at timing arc information to determine buffer/inverter)
[-verbose]          (print detailed information)

```

图 11-6 get_buffers 参数信息

输入 `get_buffers -use_arc_info`, 其结果如图 11-7 所示。

```

pt_shell> get_buffers -use_arc_info
{"SP018N_V1p0_max/PISN", "SP018N_V1p0_max/POL16N", "SP018N_V1p0_max/PISDN", "SP018N_V1p0_max/PO16N", "SP018N_V1p0_max/PO12N", "SP018N_V1p0_max/PO2N", "SP018N_V1p0_max/PINN", "SP018N_V1p0_max/PIDN", "SP018N_V1p0_max/POL8N", "SP018N_V1p0_max/PIUN", "SP018N_V1p0_max/POL12N", "SP018N_V1p0_max/PO4N", "SP018N_V1p0_max/POL24N", "SP018N_V1p0_max/PO8N", "SP018N_V1p0_max/PO24N", "SP018N_V1p0_max/PISUN", "slow/CLKBUF2", "slow/DLYX1", "slow/CLKBUF1", "slow/BUF3", "slow/BUF12", "slow/CLKBUF16", "slow/CLKBUF4", "slow/BUF2", "slow/CLKBUF8", "slow/CLKBUF1", "slow/BUF20", "slow/CLKBUFX20", "slow/CLKBUFX12", "slow/DLYX1", "slow/BUF8", "slow/CLKBUFX3", "slow/BUF1", "slow/BUF4", "slow/DLY4X1"}
pt_shell>

```

图 11-7 get_buffers -use_arc_info 命令结果

11.5 get_ports_edge_sense

该程序用于获得对应有有效沿边沿的端口信息, 其具体的代码含义如下。

```

procget_ports_edge_sense {args} {
*****
    定义名为 get_ports_edge_sense 的过程，其中包含 1 个输入参数变量 args。
*****
    parse_proc_arguments -args ${args} result
*****
    获得输入变量信息。
*****
    if { ([info exists result(-misc)] && [info exists result(-pos_edge)]) || ([info
exists result(-misc)] && [info exists result(-neg_edge)]) || ([info exists result(-
neg_edge)] && [info exists result(-pos_edge)]) } {
*****
        判断过程 get_ports_edge_sense 的必要数据输入是否完整，如果不完整则提示错误信息并返回，结束过
程 get_ports_edge_sense。
*****
        echo "Error: Specify the edge sense -neg_edge or pos_edge or -misc"
*****
        提示错误信息。
*****
        return
*****
        返回。
*****
    }
    if { [info exists result(-input)] && ![info exists result(-output)] } {
*****
        如果存在输入变量 result(-input)，则执行如下分支程序，这里人为定义为分支程序 0。分支程序 0 是对
输入端进行操作的最顶层代码，具体操作过程在分支程序 0 内再通过内部程序进行判断，并执行对应的操作。
*****
        setpos_edge_ports ""
        setneg_edge_ports ""
        setmisc_ports ""
        if { [info exists result(-neg_edge)] } {
*****
            当存在输入变量 result(-neg_edge) 时，执行如下分支程序，这里人为定义为分支程序 1。分支程序 1 是
整理出存在负边沿有效的输入端信息，并把该信息反馈给用户。
*****
            foreach_in_collection input_port [remove_from_collection [all_inputs] [get_
attribute -q [all_clocks] sources]] {
*****
                这里 get_attribute 的属性如果是 sources，说明继续！
*****
                redirect /dev/null {
                    setfanout [all_fanout -from $input_port -endpoints_only -flat -only_cells]
*****
                    all_fanout 命令是 PT 的命令，该命令是得到输出引脚所驱动的所有负载信息。
*****
                }
            }
        }
    }
}

```

```

if {$fanout!=""} {
if { ([lsearch [get_attribute -q $fanoutis_fall_edge_triggered] true] != -1) } {
append_to_collectionneg_edge_ports $input_port
}
}
}
return $neg_edge_ports
} elseif{ [info exists result(-pos_edge)] } {
*****
    当分支程序 1 判断不成立, 同时存在输入变量 result(-pos_edge) 时, 则执行如下分支程序, 这里人为定义
    为分支程序 2。分支程序 2 是整理出存在正边沿有效的输入端信息, 并把该信息反馈给用户。
    *****
    foreach_in_collectioninput_port [remove_from_collection [all_inputs] [get_
attribute -q [all_clocks ] sources]] {
    redirect /dev/null {
    setfanout [all_fanout -from $input_port -endpoints_only -flat -only_cells]
    }
    if {$fanout!=""} {
    if { ([lsearch [get_attribute -q $fanoutis_rise_edge_triggered] true] != -1) } {
    append_to_collectionpos_edge_ports $input_port
    }
    }
    }
    return $pos_edge_ports
    } elseif{ [info exists result(-misc)] } {
*****
    当分支程序 1 和分支程序 2 判断都不成立, 同时存在输入变量 result(-misc) 时, 则执行如下分支程序,
    这里人为定义为分支程序 3。分支程序 3 是整理出存在正边沿有效和存在负边沿有效的所有输入端信息, 并把该信
    息反馈给用户。
    *****
    foreach_in_collectioninput_port [remove_from_collection [all_inputs] [get_
attribute -q [all_clocks ] sources]] {
    redirect /dev/null {
    setfanout [all_fanout -from $input_port -endpoints_only -flat -only_cells]
    }
    if {$fanout==""} {
    append_to_collectionmisc_ports $input_port
    }
    if { ([lsearch [get_attribute -q $fanoutis_rise_edge_triggered] true] == -1) &&
([lsearch [get_attribute -q $fanoutis_fall_edge_triggered] true] == -1) } {
    append_to_collectionmisc_ports $input_port
    *****
    存在正边沿有效和负边沿有效的所有输入端信息, 并把值赋值给变量 misc_ports。
    *****
}
}
return $misc_ports
} else {

```



```

*****
    当分支程序 1、分支程序 2 和分支程序 3 判断都不成立时，则提示错误给用户。
    echo "Error: Specify the edge sense -neg_edge or pos_edge or -misc"
*****
}
} elseif {[info exists result(-output)] && ![info exists result(-input)] } {
*****
    结束分支程序 0，如果分支程序 0 的判断条件不成立，则进入该分支程序，这里人为定义为分支程序 4。分支程
    序 4 是对输出端进行操作的最顶层代码，具体操作过程在分支程序 4 内再通过内部程序进行判断，并执行对应的操作。
*****
    setpos_edge_ports ""
    setneg_edge_ports ""
    setmisc_ports ""
    if { [info exists result(-neg_edge)] } {
*****
        当存在输入变量 result(-neg_edge) 时，则执行如下分支程序，这里人为定义为分支程序 5。分支程序 5
        是整理出存在负边沿有效的输出端信息，并把该信息反馈给用户。
*****
        foreach_in_collection output_port [remove_from_collection [all_outputs] [get_
        attribute -q [all_clocks] sources]] {
            redirect /dev/null {
                setfanin [all_fanin -to $output_port -startpoints_only -flat -only_cells]
            }
            if {$fanin!=""} {
                if { ([lsearch [get_attribute -q $faninis_fall_edge_triggered] true] != -1) } {
                    append_to_collection neg_edge_ports $output_port
                }
            }
        }
        return $neg_edge_ports
    } elseif { [info exists result(-pos_edge)] } {
*****
        当分支程序 5 判断不成立，同时存在输入变量 result(-pos_edge) 时，则执行如下分支程序，这里人为定
        义为分支程序 6。分支程序 6 是整理出存在正边沿有效的输出端信息，并把该信息反馈给用户。
*****
        foreach_in_collection output_port [remove_from_collection [all_outputs] [get_
        attribute -q [all_clocks] sources]] {
            redirect /dev/null {
                setfanin [all_fanin -to $output_port -startpoints_only -flat -only_cells]
            }
            if {$fanin!=""} {
                if { ([lsearch [get_attribute -q $faninis_rise_edge_triggered] true] != -1) } {
                    append_to_collection pos_edge_ports $output_port
                }
            }
        }
        return $pos_edge_ports
    } elseif {[info exists result(-misc)] } {

```

当分支程序 5 和分支程序 6 判断都不成立, 同时存在输入变量 result(-misc) 时, 则执行如下分支程序, 这里人为定义为分支程序 7。分支程序 7 是整理出存在正边沿有效和负边沿有效的所有输出端信息, 并把该信息反馈给用户。

```
foreach_in_collection output_port [remove_from_collection [all_outputs] [get_attribute -q [all_clocks] sources]] {
    redirect /dev/null {
        setfanin [all_fanin -to $output_port -startpoints_only -flat -only_cells]
    }
    if {$fanin==""} {
        append_to_collection misc_ports $output_port
    }
    if { ([lsearch [get_attribute -q $faninis_rise_edge_triggered] true] == -1) &&
        ([lsearch [get_attribute -q $faninis_fall_edge_triggered] true] == -1) } {
        append_to_collection misc_ports $output_port
    }
}
return $misc_ports
} else {
```

当分支程序 5、分支程序 6 和分支程序 7 判断都不成立时, 则提示错误给用户。

```
echo "Error: Specify the edge sense -neg_edge or pos_edge or -misc"
}
} else {
```

当分支程序 0 和分支程序 4 判断都不成立时, 则提示错误给用户。

```
echo "Error: Specify the port type -input or -output"
}
```

过程 get_ports_edge_sense 结束。

```
define_proc_attributes get_ports_edge_sense \
    -define_arg{ \
        {-input      "Report Input Ports Edge Sense" "" boolean optional }

        {-output     "Report output Ports Edge Sense" "" boolean optional }
        {-neg_edge    "Report Neg_edge Sense Ports Only" "" boolean optional }
        {-pos_edge    "Report Pos_edge Sense Ports Only" "" boolean optional }
        {-misc        "Report Misc Sense Ports Only" "" boolean optional }
    }
```

PT 内嵌的名为 define_proc_attributes 的过程, 通过 define_proc_attributes 过程对输入变量 -linput、-output、-neg_edge、-pos_edge 和 -misc 进行说明定义。

输入 get_ports_edge_sense-help, 其结果如图 11-8 所示。

```
pt_shell> get_ports_edge_sense -help
Usage:
get_ports_edge_sense # Procedure
[-input]                (Report Input Ports Edge Sense)
[-output]               (Report output Ports Edge Sense)
[-neg_edge]             (Report Neg_edge Sense Ports Only)
[-pos_edge]             (Report Pos_edge Sense Ports Only)
[-misc]                (Report Misc Sense Ports Only)

pt_shell> █
```

图 11-8 get_ports_edge_sense 参数信息

输入 get_ports_edge_sense -output -pos_edge，其结果如图 11-9 所示。

```
pt_shell> get_ports_edge_sense -output -pos_edge
{"fp_cpx_req_cq[7]", "fp_cpx_req_cq[6]", "fp_cpx_req_cq[5]", "fp_cpx_req_cq[4]", "fp_cpx_req_cq[3]", "fp_cpx_req_cq[2]", "fp_cpx_req_cq[1]", "fp_cpx_req_cq[0]", "fp_cpx_data_ca[144]", "fp_cpx_data_ca[143]", "fp_cpx_data_ca[135]", "fp_cpx_data_ca[134]", "fp_cpx_data_ca[76]", "fp_cpx_data_ca[75]", "fp_cpx_data_ca[74]", "fp_cpx_data_ca[73]", "fp_cpx_data_ca[72]", "fp_cpx_data_ca[69]", "fp_cpx_data_ca[68]", "fp_cpx_data_ca[67]", "fp_cpx_data_ca[66]", "fp_cpx_data_ca[65]", "fp_cpx_data_ca[63]", "fp_cpx_data_ca[62]", "fp_cpx_data_ca[61]", "fp_cpx_data_ca[60]", "fp_cpx_data_ca[59]", "fp_cpx_data_ca[58]", "fp_cpx_data_ca[57]", "fp_cpx_data_ca[56]", "fp_cpx_data_ca[55]", "fp_cpx_data_ca[54]", "fp_cpx_data_ca[53]", "fp_cpx_data_ca[52]", "fp_cpx_data_ca[51]", "fp_cpx_data_ca[50]", "fp_cpx_data_ca[40]", "fp_cpx_data_ca[48]", "fp_cpx_data_ca[47]", "fp_cpx_data_ca[46]", "fp_cpx_data_ca[45]", "fp_cpx_data_ca[44]", "fp_cpx_data_ca[43]", "fp_cpx_data_ca[42]", "fp_cpx_data_ca[41]", "fp_cpx_data_ca[40]", "fp_cpx_data_ca[39]", "fp_cpx_data_ca[38]", "fp_cpx_data_ca[37]", "fp_cpx_data_ca[36]", "fp_cpx_data_ca[35]", "fp_cpx_data_ca[34]", "fp_cpx_data_ca[33]", "fp_cpx_data_ca[32]", "fp_cpx_data_ca[31]", "fp_cpx_data_ca[30]", "fp_cpx_data_ca[29]", "fp_cpx_data_ca[28]", "fp_cpx_data_ca[27]", "fp_cpx_data_ca[26]", "fp_cpx_data_ca[25]", "fp_cpx_data_ca[24]", "fp_cpx_data_ca[23]", "fp_cpx_data_ca[22]", "fp_cpx_data_ca[21]", "fp_cpx_data_ca[20]", "fp_cpx_data_ca[19]", "fp_cpx_data_ca[18]", "fp_cpx_data_ca[17]", "fp_cpx_data_ca[16]", "fp_cpx_data_ca[15]", "fp_cpx_data_ca[14]", "fp_cpx_data_ca[13]", "fp_cpx_data_ca[12]", "fp_cpx_data_ca[11]", "fp_cpx_data_ca[10]", "fp_cpx_data_ca[9]", "fp_cpx_data_ca[8]", "fp_cpx_data_ca[7]", "fp_cpx_data_ca[6]", "fp_cpx_data_ca[5]", "fp_cpx_data_ca[4]", "fp_cpx_data_ca[3]", "fp_cpx_data_ca[2]", "fp_cpx_data_ca[1]", "fp_cpx_data_ca[0]", "no"}
pt_shell>
```

图 11-9 get_ports_edge_sense -output -pos_edge 命令结果

11.6 report_clock_endpoint_skew

该程序用于获得对应时钟下的叶节点时序单元的时钟时序信息，其具体的代码含义如下。

```
procreport_clock_endpoint_skew { args } {
*****
  定义名为 report_clock_endpoint_skew 的过程。
*****
  globaltiming_report_unconstrained_paths
  settiming_report_unconstrained_paths true
  set results(-clock) NULL
  set results(-verbose) false
  parse_proc_arguments -args $args results
*****
  获得输入变量信息。
*****
  settargetClock $results(-clock)
  set verbose $results(-verbose)
  if { $targetClock!="ALL" } {
*****
    如果变量 targetClock 的值不等于 ALL，则执行如下分支程序。该分支程序人为定义为分支程序 0。分支
    程序 0 整理出指定时钟的信息。
*****
    redirect /dev/null {set clockList [get_clock $targetClock] }
    redirect /dev/null {add_to_collectionclockList [get_generated_clock
$targetClock] }
*****
  }
```

get_generated_clock 命令是 PT 命令, 该命令用于获得时钟信息。

```
*****
} else {
*****
```

如果分支程序 0 的判断结果不成立, 则执行如下分支程序。该分支程序人为定义为分支程序 1。分支程序 1 整理出所有的时钟信息。

```
*****
redirect /dev/null { set clockList [all_clocks] }
redirect /dev/null { add_to_collectionclockList [get_generated_clock *] }
}
foreach_in_collectioncurrentClock $clockList {
*****
```

通过命令 foreach_in_collection 依次取得变量 clockList 的值, 并循环执行。该循环程序人为定义为循环程序 0。循环程序 0 是对 clockList 进行操作的顶层代码, 具体操作过程在循环程序 0 内再通过内部程序进行判断, 并执行对应的操作。

```
*****
setClockName [get_attribute $currentClockfull_name]
echo "Info: Reporting clock skew to all sequentials clocked by $ClockName"
setprintEndPoints true
setFastestAtMax 1000
setSlowestAtMax -1000
setFastestAtMin 1000
setSlowestAtMin -1000
setregCount 0
setSlowestEndpoint "-NA-"
setFastestEndpoint "-NA-"
setclock_source [get_source_of_clock $ClockName]
*****
```

调用外部过程 get_source_of_clock, 该过程将在后面进行详细讲解。

```
*****
settmpRegisterPins [all_registers -clock $clock_source -clock_pins]
*****
```

all_registers 命令是 PT 命令, 该命令用于获得时序单元列表。

```
*****
setallRegisterPins [remove_from_collection $tmpRegisterPins [all_registers
-clock $ClockName -async]]
*****
```

```
if "[sizeof_collection $allRegisterPins] < 1" {
*****
```

如果时钟路径中不存在叶节点单元, 则返回相关信息。

```
*****
echo "Skip this clock since it has no registers attached.\n\n"
} else {
*****
```

如果时钟路径中存在叶节点单元, 则执行如下分支程序。该分支程序人为定义为分支程序 2。分支程序 2 整理出相关的时钟偏斜信息。

```
*****
setregCount [sizeof_collection $allRegisterPins]
echo "Info: There are $regCount registers on clock $ClockName."
echo "Info: looking for the slowest clock path to these registers..."
setSlowPath [get_timing_path -from $clock_source -to $allRegisterPins -delay_
*****
```

```

typemax_rise]
    setSlowestAtMax [get_attribute $SlowPath arrival]
    settmpEndpoint [get_attribute $SlowPath endpoint]
    setSlowestEndpoint [get_attribute $tmpEndpointfull_name]
    echo "Info:  $SlowestAtMax (max delay to $SlowestEndpoint).\"
    settmpPath [get_timing_path -from $ClockName -to $SlowestEndpoint -delay_
typemin_rise]
    setSlowestAtMin [get_attribute $tmpPath arrival]
    echo "Info:  $SlowestAtMin (min delay to $SlowestEndpoint).\"
    echo "Info:  looking for the fastest clock path to these registers...\"
    setMinPath [get_timing_path -from $clock_source -to $allRegisterPins -delay_
typemin_rise]
    settmpEndpoint [get_attribute $MinPath endpoint]
    setFastestAtMin [get_attribute $MinPath arrival]
    setFastestEndpoint [get_attribute $tmpEndpointfull_name]
    echo "Info:  $FastestAtMin (min delay to $FastestEndpoint).\"
    setFastPath [get_timing_path -from $ClockName -to $FastestEndpoint -delay_
typemax_rise]
    setFastestAtMax [get_attribute $FastPath arrival]
    echo "Info:  $FastestAtMax (max delay to $FastestEndpoint).\"
    echo [format \"\\n\"]
    echo [format \"      Endpoint rising edge skew report for %s\" $ClockName]
    echo [format \"      -----\"]
    echo [format \"      %s      endpoints :    %-6d\" $ClockName $regCount]
    echo [format \"      %s      skew @ max :    %-4.3f\" $ClockName [expr $SlowestAtMax -
$FastestAtMax]]
    echo [format \"      %s slowest @ max :    %-4.3f    (%s)\" $ClockName $SlowestAtMax
$SlowestEndpoint]
    echo [format \"      %s fastest @ max :    %-4.3f    (%s)\" $ClockName $FastestAtMax
$FastestEndpoint]
    echo [format \"\\n\"]
    echo [format \"      %s      skew @ min :    %-4.3f\" $ClockName [expr $SlowestAtMin -
$FastestAtMin]]
    echo [format \"      %s slowest @ min :    %-4.3f    (%s)\" $ClockName $SlowestAtMin
$SlowestEndpoint]
    echo [format \"      %s fastest @ min :    %-4.3f    (%s)\" $ClockName $FastestAtMin
$FastestEndpoint]
    echo [format \"\\n\"]
    echo [format \"\\n\"]
    echo \"Detailed path report for $ClockName tree...\"
    echo \"      To rising edge at $SlowestEndpoint\"
    echo \"      -----\"
    foreach_in_collectioncurrent_point [get_attribute $SlowPath points] {
    echo [format \"      arrival at point %-50s : %-4.3f\" \\
[get_attribute [get_attribute $current_point object] full_name] \\
[get_attribute $current_point arrival]]
    }
    echo \"\"
    echo \"      To rising edge at $FastestEndpoint\"

```

```

echo " -----"
foreach_in_collection current_point [get_attribute $FastPath points] {
echo [format "      arrival at point %-50s : %-4.3f" \
[get_attribute [get_attribute $current_point object] full_name] \
[get_attribute $current_point arrival]]
}
echo ""
echo ""
}
if { $verbose==1 } {
report_timing -path end -max 40000 -from $ClockName -to $allRegisterPins -nosplit
} else {
echo "      (use -v as the last argument to get a more verbose report)"
}
}
}

*****
过程 report_clock_endpoint_skew 结束。
*****
proc get_source_of_clock { clock } {
*****
定义名为 get_source_of_clock 的过程。
*****

set cobj [get_clock $clock]
set source [get_attribute $cobj sources]
set sc [get_attribute $cobj full_name]
set oc [get_attribute $cobj object_class]
set show [sizeof_collection $source]
if {[string compare $show 0] == 0} {
*****
判断是否是虚拟时钟, 如果是, 则返回 <virtual>。
*****
set virtual_source "<virtual>"
return $virtual_source
} else {
*****
如果不是虚拟时钟, 则整理出时钟源端点并返回该结果。
*****
foreach_in_collection nsource $source {
set name_source [get_attribute $nsource full_name]
return $name_source
}
}
}

*****
过程 get_source_of_clock 结束。
*****
define_proc_attributes report_clock_endpoint_skew -info "Analyze the Tpd to

```

```
various clock endpoints" \  
-define_args \  
{  
{-verbose "see detailed endpoint data"" boolean optional}  
{-clock "name of the clock to analyze (value ALL will analyze all  
clocks)" "clock_name" string required }  
}  
*****  
PT 内嵌的名为 define_proc_attributes 的过程，通过 define_proc_attributes 过程对输入变  
量 -clock 和 -verbose 进行说明定义。  
*****
```

输入 report_clock_endpoint_skew-help，其结果如图 11-10 所示。

```
pt_shell> report_clock_endpoint_skew -help  
Info: Procedure version 1.2  
Usage:  
report_clock_endpoint_skew # Analyze the Tpd to various clock endpoints  
[-verbose] (see detailed endpoint data)  
-clock clock_name (name of the clock to analyze (value ALL will analyze all clocks))  
pt_shell> █
```

图 11-10 report_clock_endpoint_skew 参数信息

输入 report_clock_endpoint_skew -clock gclk，其结果如图 11-11 所示。

```
pt_shell> report_clock_endpoint_skew -clock gclk  
Info: Procedure version 1.2  
Info: Reporting clock skew to all sequentials clocked by gclk  
Info: There are 4688 registers on clock gclk.  
Info: looking for the slowest clock path to these registers...  
Info: 52.958328 (max delay to fpu_add/fpu_add_exp_dp/i_alstg_dp_sngop/q_reg[0]/CK).  
Info: 52.956623 (min delay to fpu_add/fpu_add_exp_dp/i_alstg_dp_sngop/q_reg[0]/CK).  
Info: looking for the fastest clock path to these registers...  
Info: 0.000124 (min delay to cluster_header/IO/dbginit_repeater/lockup/so_l_reg/GN).  
Info: 0.000124 (max delay to cluster_header/IO/dbginit_repeater/lockup/so_l_reg/GN).  
  
Endpoint rising edge skew report for gclk  
-----  
gclk endpoints : 4688  
gclk skew @ max : 52.958  
gclk slowest @ max : 52.958 (fpu_add/fpu_add_exp_dp/i_alstg_dp_sngop/q_reg[0]/CK)  
gclk fastest @ max : 0.000 (cluster_header/IO/dbginit_repeater/lockup/so_l_reg/GN)  
  
gclk skew @ min : 52.956  
gclk slowest @ min : 52.957 (fpu_add/fpu_add_exp_dp/i_alstg_dp_sngop/q_reg[0]/CK)  
gclk fastest @ min : 0.000 (cluster_header/IO/dbginit_repeater/lockup/so_l_reg/GN)
```

图 11-11 report_clock_endpoint_skew -clock gclk 命令结果

11.7 report_violations

该程序用于获得时序违反路径信息，其具体的代码含义如下。

```
proc report_violations {args}{  
*****  
定义名为 report_violations 的过程。  
*****
```

```

*****
    parse_proc_arguments -args $args result
*****
    调用 PT 内嵌的名为 parse_proc_arguments 的过程, 提取主过程程序的输入变量并把输入变量的值赋值
    给过程程序中对应的局部变量 result。
*****

    if {[info exist result(-type)]}{
        set del_type $result(-type)
    } else {
        set del_type setup
    }
    if {[info exist result(-bins)]}{
        set bin_cnt $result(-bins)
    } else {
        set bin_cnt 5
    }
    set max_bin_limit 10
*****
    定义时序结果柱状图形的最大输出行数为 10
*****

    if {[expr ${bin_cnt}< 1} | [expr ${bin_cnt}> ${max_bin_limit}]]{
        echo "\nError: Invalid count specified for bins. Give a value from 1 to 10 only."
        echo "Info: Proceeding reporting with default bin count of 5.\n"
        set bin_cnt 5
    }
    if {[info exist result(-group)]}{
        set pthgrp [get_object_name [get_path_groups -quiet $result(-group)]]
    } else {
        set pthgrp "doitforall"
    }
    if {$pthgrp == ""}{
        echo "\nError: Invalid path-group name specified. '$result(-group)' is not a
        valid path group."
        echo "Info: Proceeding with reporting for all path groups. \n"
        set pthgrp "doitforall"
    }
    make_header
*****
    调用外部过程函数 make header。
*****

    if { ${del_type} == "setup" }{
        redirect -var rpt_vlt {report_constraint -all_violators -max_delay -nosplit -sig 5}
        set str "max_delay/setup"
    } elseif { ${del_type} == "hold" }{
        redirect -var rpt_vlt {report_constraint -all_violators -min_delay -nosplit -sig 5}
        set str "min_delay/hold"
    }

    set new_clk_grp 0
    set strt 0
    set vcnt 0
    set all_vclk_grp ""

```



```

foreach i [split $rpt_vlt "\n"] {
  if {[regexp "\s+${str}" $i]}{
    set new_clk_grp 1
    set strt 0
    set pgrp 0
    regexp "\s+${str}\\((\\'\\\\S+\\\\)' group\\)" $i match clk_nam
    lappend all_vclk_grp list_${clk_nam}
  } elseif {$new_clk_grp}{
    if {[regexp "\\--+" $i]}{
      set new_clk_grp 0
      set strt 1
    }
    } elseif {$strt}{
    if {[regexp "\s+\\S" $i]}{
      regexp "\s+\\S+\\s+\\(\\S+\\)" $i match slack
      lappend list_${clk_nam} $slack
    }
  }
  if (${pthgrp} == "doitforall"){
    set all_vclk_grp ${all_vclk_grp}
  } else {
    set op [lsearch ${all_vclk_grp} "list_${pthgrp}"]
    if {$op >= 0}{
      set all_vclk_grp "list_${pthgrp}"
    } else {
      echo "Info: Specified path group '${pthgrp}' does not have any '${del_type}'
violations.\n"
      set all_vclk_grp ""
    }
  }
  set vclk_grp_cnt [llength ${all_vclk_grp}]
  if (${vclk_grp_cnt} > 0){
    for {set i 0} {$i < ${vclk_grp_cnt}} {incr i}{
      set clk_grp [lindex ${all_vclk_grp} $i]
      set clk_grp_nam [string trim ${clk_grp} list_]
      set vltrs_lst [set [lindex ${all_vclk_grp} $i]]
      make_graph $clk_grp_nam $vltrs_lst ${bin_cnt}
    }
  }
}

*****
过程 report_violations 结束。
*****

proc get_version {}{
*****
  定义名为 get_version 的过程。
*****

global sh_product_version
return ${sh_product_version}

```

```

}

*****
过程 get_version 结束。
*****
proc max_violating_path_group_count {}{
*****
定义名为 max_violating_path_group_count 的过程。
*****
redirect -var rpt_vlt {report_constraint -all_violators -max_delay -nosplit -sig 5}
set cnt      [regexp -all "max_delay/setup" $rpt_vlt]
return $cnt
}

*****
过程 max_violating_path_group_count 结束。
*****
proc min_violating_path_group_count {}{
*****
定义名为 min_violating_path_group_count 的过程。
*****
redirect -var rpt_vlt {report_constraint -all_violators -min_delay -nosplit
-sig 5}
set cnt      [regexp -all "min_delay/hold" $rpt_vlt]
return $cnt
}

*****
过程 min_violating_path_group_count 结束。
*****
proc max_delay_violation_count {}{
*****
定义名为 proc max_delay_violation_count 的过程。
*****
redirect -var rpt_vlt {report_constraint -all_violators -max_delay -nosplit -sig 5}
set cnt      [regexp -all "VIOLATED" $rpt_vlt]
return $cnt
}

*****
过程 max_delay_violation_count 结束。
*****
proc min_delay_violation_count {}{
*****
定义名为 proc min_delay_violation_count 的过程。
*****
redirect -var rpt_vlt {report_constraint -all_violators -min_delay -nosplit -sig 5}
set cnt      [regexp -all "VIOLATED" $rpt_vlt]
return $cnt
}

*****
过程 min_delay_violation_count 结束。
*****
proc make_header {}{

```

 定义名为 make_header 的过程。

```
*****
set pth_grp      [get_path_groups *]
set des_nam      [get_object_name [current_design]]
set pgrp_nm      [sizeof ${pth_grp}]
set ver          [get_version]
set Mgrp_nm      [max_violating_path_group_count]
set mgrp_nm      [min_violating_path_group_count]
set M_v_cnt      [max_delay_violation_count]
set m_v_cnt      [min_delay_violation_count]
echo "*****"
echo "Report      : Violation Summary"
echo "Design      : ${des_nam}"
echo "Version     : $ver"
echo "Date        : [exec date]"
echo "*****\n"
echo "Total number of path groups in design : ${pgrp_nm}"
echo ""
echo "Total number of path groups with violations (max_delay/setup) : ${Mgrp_nm}"
echo "Total number of violations (max_delay/setup) in design : ${M_v_cnt}"
echo ""
echo "Total number of path groups with violations (min_delay/hold) : ${mgrp_nm}"
echo "Total number of violations (min_delay/hold) in design : ${m_v_cnt}"
echo "\n\n"
}
```

 过程 make_header 结束。

```
*****
proc in_range {s e lst}{
*****
  定义名为 in_range 的过程。
```

```
*****
set      cnt 0
foreach i $lst {
  if {[expr $i >= $e] && [expr $i < $s]}{
    incr cnt
  }
}
return $cnt
}
```

 过程 in_range 结束。

```
*****
proc get_step_size {wns bcnt}{
*****
  定义名为 get_step_size 的过程。
```

```
*****
set stp_size      [expr ${wns}/${bcnt}]
```

```

return ${stp_size}
}
*****
过程 get_step_size 结束。
*****
proc make_graph {grp_nam lst bcnt}{
*****
定义名为 make_graph 的过程。
*****
set clk_grp_nam ${grp_nam}
set vltrs_lst    $lst
set bins         $bcnt
set violtrs      [llength ${vltrs_lst}]
set max_vltr     [lindex ${vltrs_lst} 0]
set min_vltr     [lindex ${vltrs_lst} [expr $violtrs - 1]]
set stp_size     [get_step_size ${max_vltr} $bins]
for {set i 1} {$i < $bins} {incr i} {
set bar${i}      [expr ${stp_size} * ${i}]
}
set btot         0
for {set i 1} {$i < $bins} {incr i} {
set bv${i}       [in_range [expr ($i - 1) * ${stp_size}] [expr $i * ${stp_size}]
$vltrs_lst]
set btot         [expr $btot + [set bv${i}]]
}
set bv${bins}    [expr $violtrs - ($btot)]
set bltmp        ""
for {set i 1} {$i <= $bins} {incr i} {
lappend bltmp    [set bv${i}]
}
set blst         [lsort -dec -integer ${bltmp}]
set bmax         [lindex $blst 0]
set blen         [expr 50.0/${bmax}]
for {set i 1} {$i <= $bins} {incr i} {
set tmp          [set bv${i}]
set b${i}        [expr round((${tmp} * $blen) )]
}
echo "*****"
echo "Path Group Name    : ${clk_grp_nam}"
echo "Total Violations   : $violtrs"
echo "Worst Violator      : $max_vltr"
echo "*****\n"
echo "Graph Data:"
echo "===== "
for {set i 1} {$i < $bins} {incr i} {
puts "Violation Range for Bar ${i} : [format %.6f [expr ($i - 1) * ${stp_size}]] to [format %.6f [expr $i * ${stp_size}]] "
}
puts "Violation Range for Bar ${bins} : > [format %.6f [expr (${bins} - 1) *

```

```

${stp_size}]] "
    echo "=====\\n\\n"
    echo "Histogram of violations on '${clk_grp_nam}' (bins = ${bins})"
    echo "| (% of WNS)           (violations) -->"
    echo "V"
    for {set i 1}{ $i < $bins}{incr i}{
        if { $i == 1}{
            echo " [expr ($i - 1) * round(100/${bins})] - [expr $i * round(100/${bins})] % :
[string repeat * [set b${i}]] ([set bv${i}])"
        } else {
            echo "[expr ($i - 1) * round(100/${bins})] - [expr $i * round(100/${bins})] % :
[string repeat * [set b${i}]] ([set bv${i}])"
        }
    }
    echo "> [expr ($bins - 1) * round(100/${bins})] % : [string repeat * [set
b${bins}]] ([set bv${bins}])"
    echo "\\n\\n"
}

```

过程 make_graph 结束。

```

define_proc_attributes report_violations \
    -info "Display design violations (setup or hold) in histogram (classified as
per path-groups)" \
    -hide_body \
    -define_arg { \
        {-type "Specify the violation type to be reported (default: setup)" "delay_
type" one_of_string {optional value_help {values {setup hold}}}}
        {-bins "Specify the number of bins for histogram (default: 5)" "bin count (1-10)"
int optional}
        {-group "Specify the path group for reporting (default: all)" "path_group"
string optional}
    }

```

PT 内嵌的名为 define_proc_attributes 的过程，通过 define_proc_attributes 过程对输入变
量 -type、-bin 和 -group 进行说明定义。

输入 report_violations -help，其结果如图 11-12 所示。

```

pt_shell> report_violations -help
Usage:
report_violations  # Display design violations (setup or hold) in histogram (classified as per path-groups)
)
[-type delay_type] (Specify the violation type to be reported (default: setup):
Values: setup, hold)
[-bins bin count (1-10)] (Specify the number of bins for histogram (default: 5))
[-group path_group] (Specify the path group for reporting (default: all))

```

图 11-12 report_violations 参数信息

输入 report_violations -group gclk，其结果如图 11-13 所示。

```

pt_shell> report_violations -group gclk
*****
Report      : Violation Summary
Design      : fpu
Version     : D-2010.06-SP3-4
Date        : Sat Jan  2 05:12:01 EST 2016
*****

Total number of path groups in design : 4

Total number of path groups with violations (max_delay/setup) : 2
Total number of violations (max_delay/setup) in design : 4770

Total number of path groups with violations (min_delay/hold) : 1
Total number of violations (min_delay/hold) in design : 9

*****
Path Group Name : gclk
Total Violations : 4759
Worst Violator  : -80.10559
*****

Graph Data:
=====
Violation Range for Bar 1 : -0.000000 to -16.021118
Violation Range for Bar 2 : -16.021118 to -32.042236
Violation Range for Bar 3 : -32.042236 to -48.063354
Violation Range for Bar 4 : -48.063354 to -64.084472
Violation Range for Bar 5 : > -64.084472
=====

```

图 11-13 report_violations 命令结果

11.8 eco_fix_violations

该程序用于对时序违反路径进行自动时序优化，其具体的代码含义如下。

```

proc eco_fix_violations { args }{
*****
    定义名为 eco_fix_violations 的过程。
*****
    parse_proc_arguments -args ${args} result
*****
    调用 PT 内嵌的名为 parse_proc_arguments 的过程，提取主过程中的输入变量，并把输入变量的值赋值
    给过程程序中对应的局部变量 result。
*****
    suppress_message { CMD-041 }
    global verbose
    global sh_product_version
    global file_hndl
    global Time_Unit
    global allpgrp
    global pgrp_list
    global inplace
    global path_eval_mode
    global margin
    global calc_btnk
*****
    设置全局变量。

```

```

*****
lappend ::search_path " . "
set calc_btnk 0
set verbose [info exists result(-verbose)]
set inplace [info exists result(-in_place_only)]
set dontapply [info exists result(-dont_apply)]
*****

把当前数组变量 result 的数组元素的值赋值给对应变量的。
*****
set time_stamp [clock format [clock sec] -format "%a_%d_%b_%Y_%H_%M_%S"]
*****

获取当前时间信息，并赋值给变量 time_stamp
*****

print 1 ""
print $verbose "Verbose information output enabled..."
print $verbose ""
*****

调用 print 过程输出相关信息。
*****

if { [info exists result(-output)] } {
set file_name $result(-output)
print $verbose "ECO changes will be written out to file \"${file_name}\"..."
*****

如果设置了结果输出文件名信息，则设置输出文件名变量，并输出文件名设置信息。
*****

} else {
set file_name eco_fix_script_${time_stamp}.tcl
print $verbose "No output file specified by user... ECO changes will be written
out to file \"${file_name}\"..."
*****

如果没有设置结果输出文件名信息，则设置默认的输出文件名变量，并输出文件名设置信息。
*****

}
if { [info exists result(-dont_use_cells)] } {
set cell_list $result(-dont_use_cells)
*****

如果设置了不能使用的单元信息，则设置不能使用的单元信息变量。
*****

} else {
set cell_list ""
print $verbose "No eco_dont_use cells specified... all library cells will be
evaluated..."
*****

如果没有设置不能使用的单元信息，则设置所有单元都可以用于时序优化，并输出所设置的信息。
*****

}
if { [info exists result(-dont_size_cells)] } {
set inst_list $result(-dont_size_cells)
*****

如果设置了不能改变大小的单元信息，则设置不能改变大小的单元信息变量。
*****

```

```

    } else {
        set inst_list ""
        print $verbose "No restrictions on design instances specified... all cells will
be evaluated..."

```

如果没有设置不能改变大小的单元信息, 则设置所有单元都可以通过改变单元大小来进行时序优化, 并输出所设置的信息。

```

    }
    if { [info exists result(-path_group)] } {
        set pgrp_list $result(-path_group)
        set allpgrp 0

```

如果设置了具体优化的时序路径组, 则设置具体优化的时序路径组信息变量。

```

    } else {
        set pgrp_list ""
        set allpgrp 1
        print $verbose "No path-group restrictions... violations across all path-groups
will be analyzed..."

```

如果没有设置具体优化的时序路径组, 则设置所有时序路径为待优化路径, 并输出时序路径组设置信息。

```

    }
    if { [info exists result(-same_library)] } {
        alias get_alt_cells "get_alternative_lib_cells -current_library"

```

如果设置了特定的单元时序库, 则设置具体优化的单元时序库信息变量。

```

    } else {
        alias get_alt_cells "get_alternative_lib_cells"
        print $verbose "No same library restrictions... cells across different
libraries will be analyzed..."

```

如果没有设置特定的单元时序库, 则设置所有具单元时序库用于时序优化, 并输出单元时序库设置信息。

```

    }
    if { [info exists result(-path_eval_mode)] } {
        set path_eval_mode $result(-path_eval_mode)

```

如果设置了具体的时序优化模式, 则设置时序优化模式为对应模式。时序优化模式分为两种: cell_index 和 cell_delay。默认模式为 cell_index, 该模式定义时序优化基于时序路径的从起始点到终止点的顺序进行优化。当模式为 cell_delay 时, 定义时序优化从时序路径中延时最大的单元开始进行优化。

```

    } else {
        set path_eval_mode cell_index

```

如果没有设置具体的时序优化模式, 则设置时序优化模式为默认模式。

```

    }
    print ${verbose}"Cells in a timing path will be evaluated in \"${path_eval_

```



```

mode)\' mode..."
*****
    输出时序优化模式的设置信息。
*****
    if { [info exists result(-margin)] ){
        set margin $result(-margin)
*****
        如果设置了具体的时序优化结果裕度值，则设置对应的时序优化结果裕度值变量。
*****
    } else {
        set margin 50
*****
        如果没有设置具体的时序优化结果裕度值，则设置对应的时序优化结果裕度变量值为默认值 50。
*****
    }
    print ${verbose}"Violation fixing margin of ${margin} ps is specified..."
*****
    输出时序优化结果裕度值的设置信息。
*****
    set tunit [expr [get_attribute [current_design] time_unit_in_second] * 1e12]
    if { [expr $tunit == 1000] ){
        set Time_Unit "ns"
    } elseif { [expr $tunit == 1] ){
        set Time_Unit "ps"
    } else {
        set Time_Unit "${tunit} ps"
    }
*****
    设置时序优化的时间单位值。
*****
    set margin [expr ${margin}/${tunit}]
*****
    设置基于时间单位值的时序优化结果裕度值。
*****
    set file_hndl [open ${file_name} w]
*****
    打开一个可写的结果输出文件，并赋值给变量 file_hndl。
*****
    puts ${file_hndl}""
    puts ${file_hndl}""
    puts ${file_hndl}" ECO File generated by \'eco_fix_violations\' procedure "
    puts ${file_hndl}""
    puts ${file_hndl}" PrimeTime Version : ${sh_product_version}"
    puts ${file_hndl}" Date : [date] "
    puts ${file_hndl}""
    puts ${file_hndl}""
    puts ${file_hndl}"\n"
    print $verbose ""
    print $verbose "INFO: Applying eco_dont_use attribute on library cells... "
*****

```

将说明信息输出到结果输出文件中。

```
*****
apply_eco_dont_use ${cell_list}
*****
```

调用 apply_eco_dont_use 过程进行不能使用的单元设置。apply_eco_dont_use 过程代码在后面内容中进行讲解。

```
*****
print $verbose ""
print $verbose ""
print $verbose "INFO: Applying eco_dont_size attribute on design instances... "
apply_eco_dont_size ${inst_list}
*****
```

调用 apply_eco_dont_size 过程进行不能改变大小的单元设置。apply_eco_dont_size 过程代码在后面内容中进行讲解。

```
*****
print $verbose ""
if { ${inplace} } {
    print $verbose "INFO: Working in in-place sizing mode... solution set will be
very limited..."
    print $verbose ""
}
*****
```

如果设置了 in-place 模式来进行改变单元大小的时序优化设置, 则应用 in-place 模式。in-place 模式定义优化单元时其面积不能增加。

```
*****
}
suppress_message { RC-011 }
process_max_delay_report
*****
```

调用 process_max_delay_report 过程获得时序信息数据并进行优化。rocess_max_delay_report 过程代码在后面内容中讲解。

```
*****
suppress_message { RC-011 }
print 1      "\n"
print $verbose "\t\t\t\t\t --- done ---"
close ${file_hndl}
if { ! ${dontapply} } {
    print $verbose ""
    print $verbose "INFO: Design snapshot pre-fixing... "
    print $verbose ""
    report_analysis_coverage
}
*****
```

report_analysis_coverage 命令是 PT 工具的命令, 通过该命令可以得到大概的时序信息总结。这里通过执行 report_analysis_coverage 命令来得到时序优化前的大概时序信息总结。

```
*****
print $verbose ""
print $verbose "\t INFO: Sourcing design ECO fix file \"${file_name}\"'..."
print $verbose ""
redirect -file ${file_name}.log { source -echo -verbose ${file_name}; update_
timing }
*****
```

执行时序优化，并更新时序信息。

```
*****
print $verbose "\t INFO: Outputs logged to \"${file_name}.log\" file..."
print $verbose " "
print $verbose "INFO: Design snapshot post-fixing... "
print $verbose " "
report_analysis_coverage
*****
```

通过执行 report_analysis_coverage 命令来得到时序优化后的大概时序信息总结。

```
*****
print $verbose " "
}
array unset processed_cells
*****
```

清空时序优化时的单元数组变量信息。

```
*****
unsuppress_message { CMD-041 }
}
*****
```

过程 eco_fix_violations 结束。

```
*****
proc process_max_delay_report {}{
*****
```

定义名为 process_max_delay_report 的过程。

```
*****
global verbose
global processed_cells
global file_hdl
global allpgrp
global pgrp_list
global calc_btnk
print $verbose "INFO: Generating max_delay violation report..."
redirect -var rpt_vlt {report_constraint -all_violators -max_delay -nosplit -sig 5}
*****
```

report_constraint 命令是 PT 工具的命令，通过该命令可以得到时序违反的信息总结。这里通过执行命令 report_constraint 来得到时序违反信息总结，并存储到变量 rpt_vlt 中。

```
*****
print $verbose " "
if { $calc_btnk }{
*****
```

如果 calc_btnk 变量为 1，则进行关键性能瓶颈信息分析模式。

```
*****
print $verbose "INFO: Starting bottleneck analysis..."
report_bottleneck -cost_type path_cost > /dev/null
*****
```

report_bottleneck 命令是 PT 工具的命令，通过该命令可以得到时序违反中关键瓶颈的单元信息。这里通过执行命令 report_constraint 来得到时序违反中关键瓶颈的单元信息。

```
*****
print $verbose "INFO: Finished bottleneck analysis..."
}
*****
```

```

set str "max_delay/setup"
set new_clk_grp 0
set strt 0
set vcnt 0
array set processed_cells ""
print $verbose ""
print $verbose "INFO: Start parsing max_delay violation report..."
foreach i [split $rpt_vlt "\n"] {
*****
    循环获得变量 rpt_vlt 中的值。
*****
    if {[regexp "\s+${str}" $i]}{
*****
        匹配关键字 max_delay/setup。
*****
        set strt 0
        set pgrp 0
        regexp "\s+${str}\\((\\S+\\)' group\\)" $i match clk_nam
*****
        匹配关键字时钟域名。
*****
        print $verbose ""
        if { ${allpgrp} || [lsearch ${pgrp_list} ${clk_nam}] >= 0 ){
            set new_clk_grp 1
            if { !$verbose }{ echo ""}
            print 1 "INFO: Starting working on violations in path-group \"${clk_nam}\"..."
        } else {
            print 1 "INFO: Skipping violations in path-group \"${clk_nam}\"..."
        }
        } elseif { $new_clk_grp }{
            if {[regexp "\\-+" $i]}{
*****
                匹配关键字 --。
*****
                set new_clk_grp 0
                set strt 1
            }
        } elseif { $strt }{
*****
            当变量 strt 值为 1 时执行下面的代码。
*****
            if {[regexp "\\s+\\S" $i]}{
                regexp "\\s+\\(\\S+\\)\\s+\\(\\S+\\)" $i match endpnt slack
*****
                匹配关键字单元名称和时序违反值。
*****
                print $verbose ""
                print $verbose "\t...working on violating end-point \"${endpnt}\"..."
                puts ${file_hndl} "\n ECO operations for end-point ${endpnt}"
                if { !$verbose }{ echo -n "."; incr vcnt }
            }
        }
    }
}

```

```

process_max_delay_endpoint $endpnt $clk_nam
*****
    调用 process_max_delay_endpoint 过程获得时序路径中延时最大的单元信息, process_max_
delay_report 过程代码在后面内容中进行讲解。
*****
}
}
}
}
*****
    过程 process_max_delay_report 结束。
*****
proc process_max_delay_endpoint { ep clk }{
*****
    定义名为 process_max_delay_endpoint 的过程。
*****
    global verbose
    global file_hndl
    global processed_cells
    global path_eval_mode
    global margin
    global calc_btnk
    set i 0
    set flag 0
    set path_soln ""
    if { [string match async_default $clk] || [string match clock_gating_default
$clk] || [string match default $clk] ){
*****
        如果能够匹配到时钟域名称, 则执行下面代码。
*****
        set path [get_timing_paths -to $ep -group \\\*\\\\*\\$clk\\\\*\\\\*]
*****
        通过执行命令 get_timing_paths 来得到时序违反信息, 并存储到变量 path 中。
*****
    } else {
        set path [get_timing_paths -to $ep -group $clk]
*****
        通过执行命令 get_timing_paths 来得到时序违反信息, 并存储到变量 path 中, 不同的是时钟域名称。
*****
    }
    set end_pnt [get_object_name [get_attribute $path endpoint]]
    set strt_pnt [get_object_name [get_attribute $path startpoint]]
    set src_ltny [get_attribute -quiet $path startpoint_clock_latency]
    set end_ltny [get_attribute -quiet $path endpoint_clock_latency]
    set path_arrv [get_attribute -quiet $path arrival]
    set path_slack [get_attribute -quiet $path slack]
    set path_delay [expr ${path_arrv} - ${src_ltny}]
*****
    通过命令 get_attribute 获得时序路径不同类型的信息, 并存储到对应变量中。
*****
    set path_cells ""

```

```

set path_profile ""
foreach_in_collection point [get_attribute $path points] {
*****
    循环获得时序路径的节点单元。
*****

    set object      [get_attribute -quiet $point object]
    set end_trans    [get_attribute -quiet $point rise_fall]
    set port         [get_ports      -quiet $object]
    set pin          [get_pins       -quiet $object -filter "is_hierarchical==false"]
    set pin_dir      [get_attribute -quiet $pin pin_direction]
    if { $flag }{
*****
        如果变量 flag 的值为 1, 则执行下面代码。
*****

        set out_arrv [get_attribute -quiet $point arrival]
        set cell_dly  [expr ${out_arrv} - ${in_arrv}]
        set out_pin   [get_attribute -quiet [get_lib_pins -of $pin] base_name]
        set cell_cap   [get_cap $pin]
*****
        调用 cell_cap 过程来获得单元相关的负载信息。cell_cap 过程代码在后面内容中进行讲解。
*****

        set cell_nam [get_object_name [get_cells -quiet -of $pin]]
        if { [get_attribute -quiet [get_cells $cell_nam] eco_dont_size] == false }{
*****
            如果单元属性没有设置为不使用属性, 则执行下面代码。
*****

            lappend path_profile "${cell_nam} ${cell_dly} ${in_pin} ${out_pin} ${in_rtran}
            ${in_ftran} ${cell_cap}"
*****
            将单元不同的时序相关信息存储到变量 path_profile 中。
*****

        }
        set flag 0
    }
    if { [string equal ${pin_dir} in] ){
*****
        如果变量 pin_dir 的值为 in, 则执行下面代码。
*****

        set in_arrv [get_attribute -quiet ${point} arrival]
        set in_pin   [get_attribute -quiet [get_lib_pins -of $pin] base_name]
        set in_rtran [get_attribute -quiet ${pin} actual_rise_transition_max]
        set in_ftran [get_attribute -quiet ${pin} actual_fall_transition_max]
*****
        将单元引脚的不同的时序相关信息存储到对应变量中。
*****

        set flag 1
    }
    set cell [get_cells -quiet -of ${pin} -filter "is_hierarchical==false && is_
combinational==true"]
    if { [sizeof ${cell}] > 0 }{

```

```

append_to_collection -unique path_cells ${cell}
}
}

set endtrans [lindex [split ${end_trans}{}] 0]
set lvl_o_l [sizeof ${path_cells}]
set avg_dly [format %.5f [expr ${path_delay} / (1 + ${lvl_o_l})]]
print ${verbose}" "
print ${verbose}"\t\t----- End-point Stats ----- "
print ${verbose}"\t\tLevels of logic : ${lvl_o_l}"
print ${verbose}"\t\tTotal Path delay : ${path_delay}"
print ${verbose}"\t\tPath Slack : ${path_slack} (${endtrans})"
print ${verbose}"\t\t----- "

*****
输出与时序相关的说明信息。
*****
if { [string match ${path_eval_mode} cell_delay] }{
*****
如果 path_eval_mode 的值为 cell_delay, 则执行下面代码。
*****
set sorted_path_profile [lsort -decreasing -real -index 1 ${path_profile}]
*****
按递减的顺序排序变量 path_profile 的值, 并赋值给变量 sorted_path_profile。
*****
} elseif { [string match ${path_eval_mode} cell_index] }{
*****
如果 path_eval_mode 的值为 cell_index, 则执行下面代码。
*****
set sorted_path_profile ${path_profile}
*****
将变量 path_profile 的值直接赋值给变量 sorted_path_profile。
*****
}

set path_aim [expr -(${path_slack})]
set safe_lmt [expr ${margin} + ${path_aim}]
set dgain 0
set fixed 0
foreach elm ${sorted_path_profile}{
*****
循环获得变量 sorted_path_profile 的值。
*****
if {!${fixed}}{
set cell_name [lindex $elm 0]
set cell_dely [lindex $elm 1]
set cell_fpin [lindex $elm 2]
set cell_tpin [lindex $elm 3]
set cell_rtrn [lindex $elm 4]
set cell_ftrn [lindex $elm 5]
set cell_ocap [lindex $elm 6]
set already_processed [info exists processed_cells(${cell_name})]
if {!${already_processed}}{

```

```

*****
    如果变量 already_processed 的值为 0, 则执行下面代码。
*****
    set cell_area [get_attribute -quiet [get_cells ${cell_name}] area]
    set ref_cell [get_object_name [get_lib_cells -of [get_cells ${cell_name}]]]
    set full_lib [get_attribute -quiet [get_libs -of [get_lib_cells -of [get_cells
${cell_name}]]] extended_name]
    set rcel_name [file tail ${ref_cell}]
    set full_rcel ${full_lib}/${rcel_name}
*****
    将单元的不同的时序相关信息存储到对应变量中。
*****

    if { $calc_btnk }{
        set btk_cost [get_attribute -quiet [get_cells ${cell_name}] bottleneck_cost]
        if { ${btk_cost} == "" }{
            print $verbose "\t\tBottleneck cost missing for cell \'${cell_name}\'...
calculating..."
            report_bottleneck -cost path_cost -through ${cell_name}> /dev/null
            set btk_cost [get_attribute -quiet [get_cells ${cell_name}] bottleneck_cost]
        }
        if { ${btk_cost} == "" }{
            print $verbose "\t\tCannot calculate bottleneck cost for cell \'${cell_
name}\'... assigning path slack value..."
            set btk_cost [expr -(${path_slack})]
        }
        } else {
            set btk_cost 1
        }
        set cell_stat [get_lib_cell_delay ${full_rcel} ${cell_fpin} ${cell_tpin} ${cell_
rtrn} ${cell_ftrn} ${cell_ocap}]
        set cell_rdly [lindex ${cell_stat} 0]
        set cell_fdly [lindex ${cell_stat} 1]
        set cell_soln [get_solutions ${cell_name} ${cell_area} ${btk_cost} ${cell_
rdly} ${cell_fdly} \
${cell_fpin} ${cell_tpin} ${cell_rtrn} ${cell_ftrn} ${cell_ocap} ]
        print ${verbose} "\t\tFound [llength ${cell_soln}] solutions for cell \'${cell_
name}\'(${rcel_name})\'..."
*****
        调用 get_solutions 过程对时序路径进行时序优化, get_solutions 过程代码在后面内容中进行讲解。
*****

        set best_soln [lindex ${cell_soln} 0]
        if { [llength ${cell_soln}] > 0 && [lindex ${best_soln} 1] == 0 }{
            print ${verbose} "\t\t\t using solution ([file tail [lindex ${best_soln} 3]])
with cost [lindex ${best_soln} 0] and area penalty of [lindex ${best_soln} 1]..."
            set dgain [expr ${dgain} + [lindex ${best_soln} 2]]
            incr i
            puts ${file_hndl} "size_cell ${cell_name} [lindex ${best_soln} 3]"
*****
        将时序优化结果写入输出文件中。
*****

```



```

set processed_cells(${cell_name}) [lindex ${best_soln} 2]
}
} elseif (${already_processed}){
*****
    如果变量 already_processed 的值为 1，则执行下面代码。
*****
    set dgain [expr ${dgain} + $processed_cells(${cell_name})]
    incr i
    print ${verbose}"\t\t\tcell \"${cell_name}\" has already been upsized in some
previous iteration..."
}
if { ${dgain}> ${safe_lmt}}{
    set fixed 1
}
}
}
if { ${dgain} != 0 && ${dgain}>= ${safe_lmt}}{
    print ${verbose}"
    print ${verbose}"\t\t\t... total delay gain (${dgain}) achieved by sizing ${i}
cells out of total ${lvl_o_1} cells ..."
*****
    输出与时序相关的说明信息。
*****
    print ${verbose}"\t\t\t... sizing operations are estimated to be sufficient to
FIX violation of ${path_slack}..."
*****
    输出与时序相关的说明信息。
*****
} elseif { ${dgain} != 0 }{
    print ${verbose}"
    print ${verbose}"\t\t\t... total delay gain (${dgain}) by sizing operations MAY
NOT be sufficient to fix the violation..."
*****
    输出与时序相关的说明信息。
*****
}
}
*****
    过程 process_max_delay_endpoint 结束。
*****
proc get_solutions { cell area cost rdly fdly fpin tpin rtn frtn ocap }{
*****
    定义名为 get_solutions 的过程。
*****
global verbose
global Time_Unit
global inplace
set acell_soltn ""
set a 1000.0
set b 1.0

```

```

set d 1.0
redirect /dev/null { set alt_cells [filter_collection [get_alt_cells ${cell}]]
"eco_dont_use!=true"] }
foreach_in_collection alt_cell ${alt_cells}{
*****
    循环获得变量 alt_cells 的值。
*****
    set acell_area [get_attribute -quiet ${alt_cell} area]
    set acell_name [get_object_name      ${alt_cell}]
    set fpin_chk   [get_object_name [get_lib_pins -quiet  ${acell_name}/${fpin}]]
    set tpin_chk   [get_object_name [get_lib_pins -quiet  ${acell_name}/${fpin}]]
    if { ${fpin_chk} != "" && ${tpin_chk} != "" }{
        set acell_stat [get_lib_cell_delay  ${acell_name} ${fpin} ${tpin} ${rtrn}
${ftrn} ${ocap}]
*****
        调用 get_lib_cell_delay 过程得到单元的相关时序信息, get_lib_cell_delay 过程代码在后面内容
        中讲解。
*****
        set acell_rdly [lindex ${acell_stat} 0]
        set acell_fdly [lindex ${acell_stat} 1]
        if { ${acell_rdly} < ${rdly} && ${acell_fdly} < ${fdly} }{
*****
            如果替换的单元的时序结果优于原来的单元, 则执行下面代码。
*****
            set area_cost [format %.6f [expr  ${acell_area} - ${area}]]
*****
            获得替换后单元的面积开销信息, 并存储到变量 area_cost 中。
*****
            set rdly_gain [format %.6f [expr  ${rdly} - ${acell_rdly}]]
*****
            获得替换后单元的上拉延时增益信息, 并存储到变量 rdly_gain 中。
*****
            set fdly_gain [format %.6f [expr  ${fdly} - ${acell_fdly}]]
*****
            获得替换后单元的下拉延时增益信息, 并存储到变量 fdly_gain 中。
*****
            set dly_gain [format %.6f [expr  (${rdly_gain} + ${fdly_gain})/ 2.0]]
*****
            获得替换后单元的平均延时增益信息, 并存储到变量 dly_gain 中。
*****
            set totl_cost [format %.6f [expr  (${a}*${area_cost}) + 1/(${b}*${cost}) + 1/
(${d}*${dly_gain})]]
*****
            获得替换后单元的总增益信息, 并存储到变量 totl_cost 中。
*****
            if { (${inplace} && [expr ${area_cost} == 0]) || !${inplace} }{
*****
                如果面积开销满足要求, 则执行下面代码。
*****
                lappend acell_soltn "${totl_cost} ${area_cost} ${dly_gain} ${acell_name} ${rdly_
gain} ${fdly_gain}"

```

```

*****
将替换的单元相关时序信息存储到变量 acell_soltn 中。
*****
}
}
}
}
set sorted_acell_soltn [lsort -increasing -real -index 0 ${acell_soltn}]
if { ${verbose} && [llength ${sorted_acell_soltn}] > 0 }{
set fielda [string length "Alternative Cell"]
set fieldb [string length "Solution Cost"]
set fieldc [string length "Area Penalty"]
set fieldd [string length "Delay Gain (in ${Time_Unit})"]
foreach elm ${sorted_acell_soltn}{
set fielda [max ${fielda} [string length [lindex ${elm} 3]]]
set fieldb [max ${fieldb} [string length [lindex ${elm} 0]]]
set fieldc [max ${fieldc} [string length [lindex ${elm} 1]]]
set fieldd [max ${fieldd} [expr 7 + (2*[string length [lindex ${elm} 2]])]]
*****
将替换的单元相关的不同信息存储到对应变量中。
*****
}
print ${verbose}"
print ${verbose}"\t\t** Solution Table for \"${cell}\" **"
print ${verbose}"\t\t[string repeat {-} [expr ${fielda} + ${fieldb} + ${fieldc}
+ ${fieldd} + 11]]"
print ${verbose}"\t\t[format " %-*s | %-*s | %-*s | %-*s " ${fielda}"Alternative
Cell" ${fieldb}"Solution Cost" ${fieldc}"Area Penalty" ${fieldd}"Delay Gain. (in
${Time_Unit})" ]"
print ${verbose}"\t\t[string repeat {-} [expr ${fielda} + ${fieldb} + ${fieldc}
+ ${fieldd} + 11]]"
foreach sol ${sorted_acell_soltn}{
print ${verbose}"\t\t[format " %-*s | %*f | %*f | %-*s " ${fielda} [lindex $sol
3] ${fieldb} [lindex $sol 0] ${fieldc} [lindex $sol 1] ${fieldd} [lindex $sol 4](r)/
[lindex $sol 5](f) ]"
}
print ${verbose}"\t\t[string repeat {-} [expr ${fielda} + ${fieldb} + ${fieldc}
+ ${fieldd} + 11]]"
}
*****
输出与时序相关的说明信息。
*****
return ${sorted_acell_soltn}
*****
将变量 sorted_acell_soltn 的值返回调用的程序。
*****
}
*****
过程 get_solutions 结束。
*****
proc max { x y }{

```

```

*****
    定义名为 max 的过程。
*****
    if { $x > $y } { return $x } else { return $y }
*****
    比较两个变量的大小，并返回最大的值。
*****
}
*****
    过程 max 结束。
*****
proc min {`x y }{
*****
    定义名为 min 的过程。
*****
    if { $x < $y } { return $x } else { return $y }
*****
    比较两个变量的大小，并返回最小的值。
*****
}
*****
    过程 min 结束。
*****
proc get_lib_cell_delay { lib_cell fpin tpin rtran ftran cap }{
*****
    定义名为 get_lib_cell_delay 的过程。
*****
    set lcell [get_lib_cells -quiet ${lib_cell}]
    if {[sizeof $lcell]==0}{ set lcell [get_lib_cells -quiet [lindex [split ${lib_
cell} :] 1]] }
    if {[sizeof $lcell]==1}{
        redirect -variable drvprt {report_driver_model -rise_slew $rtran -fall_slew
$ftran \
        -capacitance $cap -from_pin $fpin -to_pin $tpin -lib_cell ${lcell}}
        report_driver_model 命令是 PT 工具的命令，通过该命令可以得到单元的时序信息。这里通过执行命令
report_driver_model 命令来得到对应单元的时序信息。
*****
        regexp { delay += +(.*?) +(.*?) .*} ${drvprt} match rdelay fdelay
        return "$rdelay $fdelay"
*****
        将得到的单元时序信息返回给调用的程序。
*****
    } else {
        return "1000000 1000000"
*****
        如果未得到单元的时序信息，则返回默认值给调用的程序。
*****
    }
}

```

```

*****
过程 get_lib_cell_delay 结束。
*****
proc get_cap { pin }{
*****
定义名为 get_cap 的过程。
*****
global verbose
set cap [get_attribute -quiet ${pin} effective_capacitance_max]
*****
获得单元引脚的负载信息。
*****
if { ${cap} == "" }{
set cap [get_attribute -quiet [get_nets -of ${pin}] total_capacitance_max]
*****
获得互连线的负载信息。
*****
print ${verbose}"\t\tWarning: Using lumped total cap of (${cap}) on [get_
object_name [get_cells -of ${pin}]] cell... "
}
return ${cap}
*****
返回负载值给调用的程序。
*****
}
*****
过程 get_cap 结束。
*****
proc print { ver msg }{
*****
定义名为 print 的过程。
*****
if { $ver && [regexp {^[\t]*INFO} ${msg} match] ){
echo "$msg    ([cputime], [mem] KB)"
*****
输出相关的说明信息。
*****
} elseif { $ver }{
echo "$msg"
*****
输出相关的说明信息。
*****
}
}
*****
过程 print 结束。
*****
proc apply_eco_dont_use { refcell_list }{
*****

```

定义名为 `apply_eco_dont_use` 的过程。

```
*****
redirect /dev/null { define_user_attribute -classes lib_cell eco_dont_use -type
boolean }
```

`define_user_attribute` 命令是 PT 工具中最常用的命令，它用于设置属性。这里通过 `define_user_attribute` 定义名为 `eco_dont_use` 的属性。

```
*****
foreach_in_collection lib [get_libs *] {
set lib_name [get_object_name ${lib}]
redirect /dev/null { set_user_attribute -class lib_cell [get_lib_cells ${lib_
name}/*] eco_dont_use false }
```

`set_user_attribute` 命令是 PT 工具中最常用的命令，它用于使能属性有效。这里通过 `define_user_attribute` 定义所有单元的 `eco_dont_use` 属性无效。

```
*****
foreach refcells ${refcell_list}{
set RefCells [file tail ${refcells}]
redirect /dev/null { set_user_attribute -class lib_cell [get_lib_cells -quiet
${lib_name}/${RefCells}] eco_dont_use true }
```

通过 `define_user_attribute` 定义对应单元的 `eco_dont_use` 属性有效。

```
*****
}
}
}
```

过程 `apply_eco_dont_use` 结束。

```
*****
proc apply_eco_dont_size { instance_list }{
*****
```

定义名为 `apply_eco_dont_size` 的过程。

```
*****
redirect /dev/null { define_user_attribute -classes cell eco_dont_size -type
boolean }
```

通过 `define_user_attribute` 定义名为 `eco_dont_size` 的属性。

```
*****
redirect /dev/null { set_user_attribute -class cell [get_cells -hier * -filter
"is_hierarchical==false"] eco_dont_size false }
```

通过 `define_user_attribute` 定义所有单元的 `eco_dont_size` 属性无效。

```
*****
redirect /dev/null { set_user_attribute -class cell [get_cells ${instance_
list}] eco_dont_size true }
```

通过 `define_user_attribute` 定义对应单元的 `eco_dont_size` 属性有效。

```
*****
}
*****
```

过程 apply_eco_dont_size 结束。

```
*****
define_proc_attributes eco_fix_violations \
-info "Fix setup violations with possible in-place sizing opportunities"\
-hide_body \
-define_arg {\
  {-max_delay          "Work to fix max_delay (setup) violations in design  (ON
by default)"" boolean optional }
  {-path_group         "Work on max_delay violations from specified path groups
only""path-groups"    list optional }
  {-in_place_only      "Limit ECO operations to strictly in-place sizing only (same
area)"" boolean optional }
  {-dont_use_cells     "Specify the list of lib_cells not to be used as replacement
cell""cells"          list optional }
  {-dont_size_cells    "Specify the list of design instances to be excluded from
analysis""cells"      list optional }
  {-same_library       "Limit the alternative cell search to same library only""
boolean optional }
  {-dont_apply         "Do not apply generated ECO fixes on design"" boolean
optional }
  {-path_eval_mode     "Specify the mode for evaluating cells in a timing path
(default: cell_index)""mode"  one_of_string {optional value_help {values {cell_
delay cell_index}}}}
  {-margin             "Absolute margin to which violations are fixed (default: 50
ps)""value"           int optional }
  {-verbose            "Print detailed information"" boolean optional }
  {-output             "Output file name to write out changes made""FileName"
string optional }
}
*****

PT 内嵌的名为 define_proc_attributes 的过程，通过 define_proc_attributes 的过程对输入变
量 -max_delay、-path_group、-in_palce_only、-dont_use_cells、-dont_size_cells、-same_
library、-dont_apply、-path_eval_mode、-margin、-verbose 和 -output 进行说明定义。
*****
```

输入 eco_fix_violations -help，其结果如图 11-14 所示。

```
pt_shell> eco_fix_violations -help
Usage:
eco_fix_violations  # Fix setup violations with possible in-place sizing opportunities
[-max_delay]        (Work to fix max_delay (setup) violations in design  (ON by default))
[-path_group path-groups]
                    (Work on max_delay violations from specified path groups only)
[-in_place_only]    (Limit ECO operations to strictly in-place sizing only (same area))
[-dont_use_cells cells]
                    (Specify the list of lib_cells not to be used as replacement cell)
[-dont_size_cells cells]
                    (Specify the list of design instances to be excluded from analysis)
[-same_library]     (Limit the alternative cell search to same library only)
[-dont_apply]       (Do not apply generated ECO fixes on design)
[-path_eval_mode mode]
                    (Specify the mode for evaluating cells in a timing path (default: cell_index):
                    Values: cell_delay, cell_index)
[-margin value]     (Absolute margin to which violations are fixed (default: 50 ps))
[-verbose]          (Print detailed information)
[-output FileName]  (Output file name to write out changes made)
```

图 11-14 eco_fix_violations 参数信息

输入 `eco_fix_violations -path_group gclk -output a.txt`, 其输出文件结果如图 11-15 所示。

```
#####
##
## ECO File generated by 'eco_fix_violations' procedure
##
## PrimeTime Version : D-2010.06-SP3-4
## Date : Fri Jan 1 08:18:29 2016
##
#####

## ECO operations for end-point fpu_add/fpu_add_frac_dp/i_alstg_inla/q_reg[0]/D
## ECO operations for end-point fpu_add/fpu_add_frac_dp/i_alstg_inla/q_reg[1]/D
## ECO operations for end-point fpu_add/fpu_add_frac_dp/i_alstg_inla/q_reg[2]/D
## ECO operations for end-point fpu_add/fpu_add_frac_dp/i_alstg_inla/q_reg[3]/D
## ECO operations for end-point fpu_add/fpu_add_frac_dp/i_alstg_inla/q_reg[4]/D
## ECO operations for end-point fpu_add/fpu_add_frac_dp/i_alstg_inla/q_reg[5]/D
```

图 11-15 `eco_fix_violations -path_group gclk -output a.txt` 命令输出文件结果

附 录

1. NLDM 时序信息文件片段

```
library(typical) {  
  
    /* general attributes */  
    delay_model : table_lookup;  
    in_place_swap_mode : match_footprint;  
    library_features(report_delay_calculation);  
  
    /* documentation attributes */  
    revision : 1.1;  
    date : "Wed Mar 10 10:48:51 2016";  
    comment : "www.icdream.com";  
  
    /* unit attributes */  
    time_unit : "1ns";  
    voltage_unit : "1V";  
    current_unit : "1mA";  
    pulling_resistance_unit : "1kohm";  
    leakage_power_unit : "1pW";  
    capacitive_load_unit (1.0,pf);  
  
    /* operation conditions */  
    nom_process : 1;  
    nom_temperature : 125;  
    nom_voltage : 1.08;  
    operating_conditions(slow) {  
        process : 1;  
        temperature : 125;  
        voltage : 1.08;  
        tree_type : balanced_tree  
    }  
    default_operating_conditions : slow;  
  
    /* threshold definitions */  
    slew_lower_threshold_pct_fall : 30.0;  
    slew_upper_threshold_pct_fall : 70.0;  
    slew_lower_threshold_pct_rise : 30.0;  
    slew_upper_threshold_pct_rise : 70.0;  
    input_threshold_pct_fall : 50.0;
```

```

input_threshold_pct_rise : 50.0;
output_threshold_pct_fall : 50.0;
output_threshold_pct_rise : 50.0;
slew_derate_from_library : 0.5;

/* default attributes */
default_leakage_power_density : 0.0;
default_cell_leakage_power : 0.0;
default_fanout_load : 1.0;
default_output_pin_cap : 0.0;
default_inout_pin_cap : 0.00158;
default_input_pin_cap : 0.00158;
default_max_transition : 1.5;

/* templates */
lu_table_template(delay_template_7x1) {
    variable_1 : input_net_transition;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
power_lut_template(energy_template_7x1) {
    variable_1 : input_transition_time;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
lu_table_template(delay_template_7x7) {
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
    index_2 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
power_lut_template(energy_template_7x7) {
    variable_1 : input_transition_time;
    variable_2 : total_output_net_capacitance;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
    index_2 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
power_lut_template(energy_template_1x7) {
    variable_1 : total_output_net_capacitance;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
power_lut_template(energy_template_7x3x3) {
    variable_1 : input_transition_time;
    variable_2 : total_output_net_capacitance;
    variable_3 : equal_or_opposite_output_net_capacitance;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
    index_2 ("1000, 1001, 1002");
    index_3 ("1000, 1001, 1002");
}
power_lut_template(passive_energy_template_1x7) {
    variable_1 : input_transition_time;
    index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}

```

```

lu_table_template(setup_template_3x3) {
    variable_1 : constrained_pin_transition;
    variable_2 : related_pin_transition;
    index_1 ("1000, 1001, 1002");
    index_2 ("1000, 1001, 1002");
}

lu_table_template(hold_template_3x3) {
    variable_1 : constrained_pin_transition;
    variable_2 : related_pin_transition;
    index_1 ("1000, 1001, 1002");
    index_2 ("1000, 1001, 1002");
}

lu_table_template(recovery_template_3x3) {
    variable_1 : constrained_pin_transition;
    variable_2 : related_pin_transition;
    index_1 ("1000, 1001, 1002");
    index_2 ("1000, 1001, 1002");
}

lu_table_template(removal_template_3x3) {
    variable_1 : constrained_pin_transition;
    variable_2 : related_pin_transition;
    index_1 ("1000, 1001, 1002");
    index_2 ("1000, 1001, 1002");
}

/* k-factors */
k_process_cell_leakage_power : 0;
k_temp_cell_leakage_power : 0;
k_volt_cell_leakage_power : 0;
k_process_internal_power : 0;
k_temp_internal_power : 0;
k_volt_internal_power : 0;
k_process_rise_transition : 1;
k_temp_rise_transition : 0;
k_volt_rise_transition : 0;
k_process_fall_transition : 1;
k_temp_fall_transition : 0;
k_volt_fall_transition : 0;
k_process_setup_rise : 1;
k_temp_setup_rise : 0;
k_volt_setup_rise : 0;
k_process_setup_fall : 1;
k_temp_setup_fall : 0;
k_volt_setup_fall : 0;
k_process_hold_rise : 1;
k_temp_hold_rise : 0;
k_volt_hold_rise : 0;
k_process_hold_fall : 1;
k_temp_hold_fall : 0;
k_volt_hold_fall : 0;

```

```

k_process_min_pulse_width_high : 1;
k_temp_min_pulse_width_high : 0;
k_volt_min_pulse_width_high : 0;
k_process_min_pulse_width_low : 1;
k_temp_min_pulse_width_low : 0;
k_volt_min_pulse_width_low : 0;
k_process_recovery_rise : 1;
k_temp_recovery_rise : 0;
k_volt_recovery_rise : 0;
k_process_recovery_fall : 1;
k_temp_recovery_fall : 0;
k_volt_recovery_fall : 0;
k_process_cell_rise : 1;
k_temp_cell_rise : 0;
k_volt_cell_rise : 0;
k_process_cell_fall : 1;
k_temp_cell_fall : 0;
k_volt_cell_fall : 0;
k_process_wire_cap : 0;
k_temp_wire_cap : 0;
k_volt_wire_cap : 0;
k_process_wire_res : 0;
k_temp_wire_res : 0;
k_volt_wire_res : 0;
k_process_pin_cap : 0;
k_temp_pin_cap : 0;
k_volt_pin_cap : 0;

```

```

/* pad attributes */
output_voltage(GENERAL) {
    vol : 0.4;
    voh : VDD - 0.4;
    vomin : -0.5;
    vomax : VDD + 0.5;
}
input_voltage(CMOS) {
    vil : 0.3 * VDD;
    vih : 0.7 * VDD;
    vimin : -0.5;
    vimax : VDD + 0.5;
}
input_voltage(TTL) {
    vil : 0.8;
    vih : 2;
    vimin : -0.5;
    vimax : VDD + 0.5;
}

```

```

/* wire-loads */
wire_load("csml3_wl10") {

```

```

    resistance : 8.5e-8;
    capacitance : 1.5e-4;
    area : 0.7;
    slope : 66.667;
    fanout_length (1,66.667);
}
wire_load("csm13_wl20") {
    resistance : 8.5e-8;
    capacitance : 1.5e-4;
    area : 0.7;
    slope : 133.334;
    fanout_length (1,133.334);
}
wire_load("csm13_wl30") {
    resistance : 8.5e-8;
    capacitance : 1.5e-4;
    area : 0.7;
    slope : 200.001;
    fanout_length (1,200.001);
}
wire_load("csm13_wl40") {
    resistance : 8.5e-8;
    capacitance : 1.5e-4;
    area : 0.7;
    slope : 266.668;
    fanout_length (1,266.668);
}
wire_load("csm13_wl50") {
    resistance : 8.5e-8;
    capacitance : 1.5e-4;
    area : 0.7;
    slope : 333.335;
    fanout_length (1,333.335);
}
/* QA wire-load */
wire_load("ForQA") {
    resistance : 0;
    capacitance : 1;
    area : 1;
    slope : 1;
    fanout_length(1,0);
    fanout_length(10,0);
}

cell (DFFX1) {
    cell_footprint : dff;
    area : 27.158400;
    pin(D) {
        direction : input;
        capacitance : 0.001351;
    }
}

```

```

internal_power() {
    rise_power(passive_energy_template_1x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        values ("0.006664, 0.006675, 0.006701, 0.006853, 0.007333, 0.008608,
0.011514");
    }
    fall_power(passive_energy_template_1x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        values ("0.008450, 0.008416, 0.008423, 0.008550, 0.009064, 0.010399,
0.013363");
    }
}

timing() {
    related_pin : "CK";
    timing_type : setup_rising;
    rise_constraint(setup_template_3x3) {
        index_1 ("0.042, 0.392, 1.5");
        index_2 ("0.042, 0.392, 0.764");
        values ( \
            "0.074219, 0.050781, 0.058594", \
            "0.093750, 0.062500, 0.066406", \
            "0.062500, 0.023438, 0.027344");
    }
    fall_constraint(setup_template_3x3) {
        index_1 ("0.042, 0.392, 1.5");
        index_2 ("0.042, 0.392, 0.764");
        values ( \
            "0.148438, 0.101562, 0.093750", \
            "0.214844, 0.167969, 0.160156", \
            "0.421875, 0.375000, 0.367188");
    }
}

timing() {
    related_pin : "CK";
    timing_type : hold_rising;
    rise_constraint(hold_template_3x3) {
        index_1 ("0.042, 0.392, 1.5");
        index_2 ("0.042, 0.392, 0.764");
        values ( \
            "-0.046875, -0.035156, -0.042969", \
            "-0.062500, -0.042969, -0.046875", \
            "-0.015625, 0.011719, 0.003906");
    }
    fall_constraint(hold_template_3x3) {
        index_1 ("0.042, 0.392, 1.5");
        index_2 ("0.042, 0.392, 0.764");
        values ( \
            "-0.042969, -0.003906, 0.011719", \
            "-0.117188, -0.070312, -0.050781", \
            "-0.308594, -0.261719, -0.242188");
    }
}

```

```

    }
  }
}
pin(CK) {
  direction : input;
  clock : true;
  capacitance : 0.001907;
  max_transition : 0.764000;
  internal_power() {
    when : "D";
    rise_power(passive_energy_template_1x7) {
      index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
      values ("0.008465, 0.008478, 0.008525, 0.008762, 0.009477, 0.011300,
0.015396");
    }
    fall_power(passive_energy_template_1x7) {
      index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
      values ("0.009359, 0.009359, 0.009433, 0.009723, 0.010528, 0.012405,
0.016504");
    }
  }
  internal_power() {
    when : "!D";
    rise_power(passive_energy_template_1x7) {
      index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
      values ("0.009127, 0.009143, 0.009188, 0.009427, 0.010147, 0.011979,
0.016083");
    }
    fall_power(passive_energy_template_1x7) {
      index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
      values ("0.009323, 0.009326, 0.009409, 0.009712, 0.010530, 0.012424,
0.016545");
    }
  }
  min_pulse_width_high : 0.102980;
  min_pulse_width_low : 0.151562;
}
ff(IQ,IQN) {
  clocked_on : "CK";
  next_state : "D";
}
pin(Q) {
  direction : output;
  capacitance : 0.0;
  function : "IQ";
  internal_power() {
    related_pin : "CK";
    equal_or_opposite_output : "QN";
    rise_power(energy_template_7x3x3) {
      index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
    }
  }
}

```

```

        index_2 ("0.00079, 0.010112, 0.08532");
        index_3 ("0.00079, 0.010112, 0.08532");
        values ( \
            "0.007121, 0.007687, 0.007931", "0.007685, 0.007558, 0.007799",
"0.007092, 0.006932, 0.006904", \
            "0.007116, 0.007682, 0.007925", "0.007678, 0.007552, 0.007794",
"0.007086, 0.006926, 0.006898", \
            "0.007113, 0.007680, 0.007924", "0.007676, 0.007550, 0.007792",
"0.007085, 0.006924, 0.006896", \
            "0.007111, 0.007678, 0.007922", "0.007675, 0.007548, 0.007790",
"0.007082, 0.006922, 0.006894", \
            "0.007117, 0.007688, 0.007933", "0.007683, 0.007555, 0.007798",
"0.007088, 0.006928, 0.006900", \
            "0.007920, 0.007792, 0.007996", "0.007750, 0.007621, 0.007864",
"0.007153, 0.006992, 0.006964", \
            "0.008180, 0.008079, 0.008137", "0.008101, 0.007976, 0.007998",
"0.007286, 0.007125, 0.007097");
    }
    fall_power(energy_template_7x3x3) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.010112, 0.08532");
        index_3 ("0.00079, 0.010112, 0.08532");
        values ( \
            "0.008044, 0.008372, 0.007839", "0.008149, 0.008216, 0.007632",
"0.008476, 0.008390, 0.007543", \
            "0.008030, 0.008357, 0.007825", "0.008134, 0.008201, 0.007618",
"0.008461, 0.008375, 0.007528", \
            "0.008023, 0.008350, 0.007818", "0.008124, 0.008190, 0.007608",
"0.008453, 0.008367, 0.007519", \
            "0.008054, 0.008388, 0.007858", "0.008151, 0.008217, 0.007635",
"0.008482, 0.008394, 0.007545", \
            "0.008211, 0.008554, 0.008020", "0.008279, 0.008341, 0.007757",
"0.008605, 0.008506, 0.007655", \
            "0.009207, 0.009020, 0.008481", "0.008924, 0.008728, 0.008141",
"0.008963, 0.008857, 0.008001", \
            "0.010411, 0.010304, 0.009524", "0.009987, 0.009874, 0.009046",
"0.009816, 0.009683, 0.008815");
    }
}
timing() {
    related_pin : "CK";
    timing_type : rising_edge;
    timing_sense : non_unate;
    cell_rise(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \
            "0.195580, 0.203842, 0.219522, 0.248916, 0.304326, 0.411304, 0.626459", \
            "0.201038, 0.209404, 0.225007, 0.254441, 0.309886, 0.416836, 0.631982", \

```



```

        "0.211275, 0.219565, 0.235192, 0.264602, 0.320024, 0.426986, 0.642127", \
        "0.228180, 0.236392, 0.252109, 0.281526, 0.336946, 0.443925, 0.659078", \
        "0.248416, 0.256687, 0.272361, 0.301808, 0.357237, 0.464189, 0.679321", \
        "0.268028, 0.276231, 0.291977, 0.321395, 0.376847, 0.483812, 0.698926", \
        "0.276088, 0.284376, 0.300115, 0.329557, 0.385018, 0.491967, 0.707086");
    }
    rise_transition(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \
            "0.042386, 0.053160, 0.077941, 0.128840, 0.228813, 0.434041, 0.853871", \
            "0.042498, 0.053093, 0.077775, 0.129020, 0.228697, 0.434092, 0.853897", \
            "0.042825, 0.053151, 0.077868, 0.128964, 0.228724, 0.434100, 0.853878", \
            "0.041976, 0.053094, 0.077864, 0.128825, 0.228813, 0.434049, 0.853869", \
            "0.042765, 0.052895, 0.077925, 0.129021, 0.228716, 0.434102, 0.853884", \
            "0.042352, 0.053743, 0.077734, 0.129064, 0.228760, 0.434120, 0.853893", \
            "0.042505, 0.053625, 0.078257, 0.129088, 0.228893, 0.434039, 0.853901");
    }
    cell_fall(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \
            "0.138140, 0.146006, 0.160439, 0.184940, 0.225725, 0.293421, 0.419438", \
            "0.143499, 0.151438, 0.165910, 0.190400, 0.231190, 0.298880, 0.424905", \
            "0.153580, 0.161473, 0.175948, 0.200434, 0.241228, 0.308926, 0.434946", \
            "0.169696, 0.177645, 0.192010, 0.216555, 0.257369, 0.325091, 0.451131", \
            "0.187626, 0.195645, 0.210158, 0.234882, 0.275877, 0.343732, 0.469819", \
            "0.202698, 0.210855, 0.225640, 0.250763, 0.292227, 0.360442, 0.486683", \
            "0.203145, 0.211690, 0.227191, 0.253212, 0.295654, 0.364816, 0.491491");
    }
    fall_transition(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \
            "0.041457, 0.050025, 0.066430, 0.097618, 0.149141, 0.248138, 0.464151", \
            "0.041827, 0.049993, 0.066467, 0.097554, 0.149078, 0.248204, 0.464137", \
            "0.041604, 0.050017, 0.066465, 0.097552, 0.149132, 0.248189, 0.464150", \
            "0.042285, 0.050394, 0.066682, 0.097990, 0.149226, 0.248237, 0.464165", \
            "0.043570, 0.051222, 0.067715, 0.098881, 0.149864, 0.248611, 0.464248", \
            "0.045980, 0.053787, 0.070116, 0.101171, 0.151633, 0.249668, 0.464632", \
            "0.051084, 0.059152, 0.075209, 0.105841, 0.155875, 0.252454, 0.465703");
    }
}
max_capacitance : 0.085320;
}
pin(QN) {
    direction : output;

```

```

capacitance : 0.0;
function : "IQN";
timing() {
    related_pin : "CK";
    timing_type : rising_edge;
    timing_sense : non_unate;
    cell_rise(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \
            "0.198494, 0.207019, 0.222498, 0.250479, 0.304563, 0.411143, 0.626181", \
            "0.203961, 0.212472, 0.227971, 0.255972, 0.309976, 0.416604, 0.631623", \
            "0.213982, 0.222493, 0.237972, 0.265979, 0.320000, 0.426615, 0.641637", \
            "0.230013, 0.238479, 0.254072, 0.281981, 0.336080, 0.442653, 0.657683", \
            "0.248084, 0.256487, 0.272123, 0.300043, 0.354159, 0.460719, 0.675756", \
            "0.263836, 0.272330, 0.287997, 0.315932, 0.369977, 0.476515, 0.691555", \
            "0.265896, 0.274458, 0.290236, 0.318237, 0.372204, 0.478729, 0.693724");
    }
    rise_transition(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \
            "0.038407, 0.048804, 0.070340, 0.117946, 0.220651, 0.427967, 0.848415", \
            "0.038857, 0.048714, 0.070516, 0.117984, 0.220602, 0.428017, 0.848403", \
            "0.038965, 0.048828, 0.070498, 0.117917, 0.220620, 0.428007, 0.848395", \
            "0.037912, 0.048520, 0.070543, 0.118179, 0.220646, 0.427979, 0.848446", \
            "0.038089, 0.048213, 0.070752, 0.118262, 0.220688, 0.427953, 0.848436", \
            "0.038575, 0.049113, 0.071107, 0.118453, 0.220716, 0.428000, 0.848426", \
            "0.039597, 0.050310, 0.071928, 0.118860, 0.220807, 0.428077, 0.848447");
    }
    cell_fall(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \
            "0.255598, 0.262889, 0.275738, 0.296449, 0.330814, 0.391754, 0.510442", \
            "0.261051, 0.268437, 0.281290, 0.301979, 0.336324, 0.397260, 0.515959", \
            "0.271203, 0.278642, 0.291429, 0.312137, 0.346480, 0.407415, 0.526119", \
            "0.288220, 0.295517, 0.308380, 0.329090, 0.363434, 0.424372, 0.543075", \
            "0.308384, 0.315813, 0.328605, 0.349317, 0.383661, 0.444604, 0.563302", \
            "0.328119, 0.335376, 0.348278, 0.368977, 0.403337, 0.464277, 0.582965", \
            "0.336324, 0.343620, 0.356470, 0.377178, 0.411557, 0.472501, 0.591192");
    }
    fall_transition(delay_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.392, 0.764, 1.5");
        index_2 ("0.00079, 0.002054, 0.00474, 0.010112, 0.020856, 0.042186,
0.08532");
        values ( \

```

```

        "0.033791, 0.041257, 0.054058, 0.078532, 0.126808, 0.226736, 0.438425", \
        "0.034189, 0.040630, 0.054301, 0.078649, 0.126637, 0.226700, 0.438488", \
        "0.034236, 0.040819, 0.053916, 0.078777, 0.126729, 0.226646, 0.438482", \
        "0.033894, 0.040976, 0.053918, 0.078777, 0.126730, 0.226649, 0.438481", \
        "0.034242, 0.040930, 0.053940, 0.078784, 0.126741, 0.226668, 0.438478", \
        "0.033613, 0.041132, 0.054410, 0.078274, 0.126806, 0.226781, 0.438477", \
        "0.033850, 0.041292, 0.054068, 0.078580, 0.126828, 0.226746, 0.438431");
    }
}
max_capacitance : 0.085320;
}
cell_leakage_power : 104012.888400;
leakage_power() {
    when : "!D & !CK & !Q & QN";
    value : 90368.632800;
}
leakage_power() {
    when : "!D & !CK & Q & !QN";
    value : 85672.587600;
}
leakage_power() {
    when : "!D & CK & !Q & QN";
    value : 89230.410000;
}
leakage_power() {
    when : "!D & CK & Q & !QN";
    value : 79203.700800;
}
leakage_power() {
    when : "D & !CK & !Q & QN";
    value : 89305.092000;
}
leakage_power() {
    when : "D & !CK & Q & !QN";
    value : 68513.785200;
}
leakage_power() {
    when : "D & CK & !Q & QN";
    value : 104012.888400;
}
leakage_power() {
    when : "D & CK & Q & !QN";
    value : 77959.227600;
}
}
}

```

2. CCS 时序信息文件片段

```

library ("typical") {
    delay_model : "table_lookup";
}

```

```

library_features("report_delay_calculation", \ "report_user_data");
in_place_swap_mode : "match_footprint";
time_unit : "lps";
voltage_unit : "1V";
current_unit : "1mA";
pulling_resistance_unit : "1kohm";
capacitive_load_unit(1, \ "ff");
leakage_power_unit : "1mW";
default_inout_pin_cap : 999;
default_input_pin_cap : 999;
default_output_pin_cap : 0;
default_fanout_load : 1;
default_max_transition : 1000.000000;
default_leakage_power_density : 0;
default_cell_leakage_power : 0;
k_process_rise_propagation : 0;
k_process_fall_propagation : 0;
k_process_rise_transition : 0;
k_process_fall_transition : 0;
k_temp_rise_propagation : 0;
k_temp_fall_propagation : 0;
k_temp_rise_transition : 0;
k_temp_fall_transition : 0;
k_volt_rise_propagation : 0;
k_volt_fall_propagation : 0;
k_volt_rise_transition : 0;
k_volt_fall_transition : 0;
slew_lower_threshold_pct_rise : 20.000000;
slew_upper_threshold_pct_rise : 80.000000;
input_threshold_pct_fall : 50;
output_threshold_pct_fall : 50;
input_threshold_pct_rise : 50;
output_threshold_pct_rise : 50;
slew_lower_threshold_pct_fall : 20.000000;
slew_upper_threshold_pct_fall : 80.000000;
slew_derate_from_library : 1.000000;
nom_process : 1;
nom_temperature : 100.000000;
nom_voltage : 0.720000;
default_wire_load_mode : "segmented";
voltage_map("VDD",0.720000);
voltage_map("VDDST",0.720000);
voltage_map("VDDGLOBAL",0.720000);
voltage_map("VDDINT",0.720000);
voltage_map("VDDSRC",0.720000);
voltage_map("VSS",0.000000);
/* Define how delays will be calculated. */
operating_conditions (BEST) {
    process : 1;
    temperature : 100.000000;

```

```

        voltage : 0.720000;
        tree_type : "best_case_tree";
    }
    operating_conditions (BALANCED) {
        process : 1;
        temperature : 100.000000;
        voltage : 0.720000;
        tree_type : "balanced_tree";
    }
    operating_conditions (WORST) {
        process : 1;
        temperature : 100.000000;
        voltage : 0.720000;
        tree_type : "worst_case_tree";
    }
    /*      Wireload models.      */
    wire_load (DEFAULT) {
        capacitance : 0;
        resistance : 0;
        area : 0;
        slope : 0.050000;
        fanout_length(1,0.062500);
        fanout_length(2,0.187500);
        fanout_length(8,0.937500);
    }
    /*      Wire length (mm) for 8 fanouts      */
    wire_load ("ZERO_LOAD") {
        capacitance : 0;
        resistance : 0;
        area : 0;
        slope : 0.050000;
        fanout_length(1,0.062500);
        fanout_length(2,0.187500);
        fanout_length(8,0.937500);
    }
    lu_table_template ("CCS_RCV_TEMPLATE_0") {
        variable_1 : "input_net_transition";
        index_1("1, 2, 3, 4, 5, 6, 7");
        variable_2 : "total_output_net_capacitance";
        index_2("1, 2");
    }
    lu_table_template ("ccsn_dc") {
        variable_1 : "input_voltage";
        variable_2 : "output_voltage";
    }
    lu_table_template ("ccsn_ovrf") {
        variable_1 : "input_net_transition";
        variable_2 : "total_output_net_capacitance";
        variable_3 : "time";
    }
}

```

```

lu_table_template ("ccsn_pnlh") {
    variable_1 : "input_noise_height";
    variable_2 : "input_noise_width";
    variable_3 : "total_output_net_capacitance";
    variable_4 : "time";
}

lu_table_template ("del_1_7_7") {
    variable_1 : "input_net_transition";
    index_1("1, 2, 3, 4, 5, 6, 7");
    variable_2 : "total_output_net_capacitance";
    index_2("1, 2, 3, 4, 5, 6, 7");
}

lu_table_template ("driver_waveform_template__ckbf8x") {
    variable_1 : "input_net_transition";
    index_1("1, 2");
    variable_2 : "normalized_voltage";
    index_2("1, 2");
}

/*      Extra templates.      */
output_current_template ("CCS_DRV_TEMPLATE_0") {
    variable_1 : "input_net_transition";
    variable_2 : "total_output_net_capacitance";
    variable_3 : "time";
}

power_lut_template ("power_outputs_1") {
    variable_1 : "input_transition_time";
    index_1("1, 2, 3, 4, 5, 6, 7");
    variable_2 : "total_output_net_capacitance";
    index_2("1, 2, 3, 4, 5, 6, 7");
}

normalized_driver_waveform ("driver_waveform_template__ckbf8x") {
    driver_waveform_name : "preDrv__ckbf8x";
    index_1("2.078093, 4.719815, ...");
    index_2("0, 0.05, 0.2, 0.3325198, ...");
    values("0, 0.3463488, 0.6926976, ...", \
        ...
    )
}

sensitization ("sens_1in_1out") {
    pin_names("IN0", \
        "OUT0");
    vector("0", \
        "0 0");
    vector("1", \
        "1 0");
    vector("2", \
        "0 1");
    vector("3", \
        "1 1");
    vector("4", \
        "0 Z");
}

```

```

vector("5", \
    "1 Z");
vector("6", \
    "0 X");
vector("7", \
    "1 X");
}
cell (ckbf8x) {
    dont_use : true;
    cell_footprint : "ckbf";
    driver_waveform_fall : "preDrv__ckbf8x";
    driver_waveform_rise : "preDrv__ckbf8x";
    pin_name_map("CLK", \ "Z");
    sensitization_master : "sens_lin_lout";
    area : 1.797120;
    cell_leakage_power : 2.228420e-04;
    pg_pin (VDD) {
        pg_type : "primary_power";
        voltage_name : "VDD";
    }
    pg_pin (VSS) {
        pg_type : "primary_ground";
        voltage_name : "VSS";
    }
    leakage_power () {
        related_pg_pin : "VDD";
        value : 2.636000e-04;
        when : "!CLK";
    }
    leakage_power () {
        related_pg_pin : "VDD";
        value : 1.821000e-04;
        when : "CLK";
    }
    pin (CLK) {
        capacitance : 2.408479;
        direction : "input";
        fall_capacitance : 2.418984;
        fall_capacitance_range("1.977593", \
            "2.684779");
        max_transition : 168.507000;
        related_ground_pin : "VSS";
        related_power_pin : "VDD";
        rise_capacitance : 2.397974;
        rise_capacitance_range("1.921266", \
            "2.661267");
    }
    pin (Z) {
        direction : "output";
        function : "CLK";
    }
}

```

```

max_capacitance : 435.754000;
max_transition : 168.281000;
min_capacitance : 0.001000;
power_down_function : "!VDD + VSS";
related_ground_pin : "VSS";
related_power_pin : "VDD";
internal_power () {
    related_pg_pin : "VDD";
    related_pin : "CLK";
    fall_power ("power_outputs_1") {
        index_1("2.07809, 4.71982, 10.0033, 20.5701, 41.7039, 83.9715,
168.507");
        index_2("0.001, 7.817825, 23.30679, 53.99805, 114.8125, 235.3158,
474.092");
        values("4.033592, 3.881643, 3.945105, 3.998839, 4.02715, 4.037324,
4.03333", \
            "4.050711, 3.873893, 3.929798, 3.988034, 4.018997, 4.030466,
4.027836", \
            "4.141694, 3.910199, 3.940568, 3.998624, 4.036149, 4.052675,
4.052908", \
            "4.479205, 4.120108, 4.078142, 4.115624, 4.15324, 4.173082,
4.176327", \
            "5.234, 4.687286, 4.518244, 4.49781, 4.517586, 4.535908,
4.539733", \
            "6.749532, 5.97609, 5.602038, 5.465923, 5.430537, 5.427381,
5.425151", \
            "9.725376, 8.712815, 8.019273, 7.663697, 7.49798, 7.427644,
7.392512");
    }
    rise_power ("power_outputs_1") {
        index_1("2.07809, 4.71982, 10.0033, 20.5701, 41.7039, 83.9715,
168.507");
        index_2("0.001, 7.817825, 23.30679, 53.99805, 114.8125, 235.3158,
474.092");
        values("2.580916, 2.460673, 2.527192, 2.574172, 2.601021, 2.549744,
2.899463", \
            "2.601233, 2.455338, 2.51742, 2.566098, 2.600252, 2.631291,
2.809409", \
            "2.700434, 2.497634, 2.539733, 2.595084, 2.642055, 2.670829,
2.704356", \
            "3.048484, 2.71313, 2.675254, 2.714494, 2.771095, 2.837484,
2.881696", \
            "3.828445, 3.297743, 3.134774, 3.106045, 3.133119, 3.273964,
3.29091", \
            "5.422142, 4.644612, 4.286657, 4.120342, 4.083909, 4.084751,
4.372758", \
            "8.56212, 7.522587, 6.839761, 6.457066, 6.241802, 6.165368,
6.165485");
    }
}
}

```



```

/* from hold file */
timing () {
    related_pin : "CLK";
    timing_sense : "positive_unate";
    wave_fall("3", \ "0");
    wave_rise("0", \ "3");
    ccsn_first_stage () {
        is_inverting : true;
        is_needed : true;
        miller_cap_fall : 0.822435;
        miller_cap_rise : 0.788123;
        stage_type : "both";
        dc_current ("ccsn_dc") {
            index_1("-0.72, -0.36, ...");
            index_2("-0.72, -0.36, ...");
            values("1.71046, 1.61329, ...", \
                .....
            )
        }
        output_voltage_fall () {
            vector ("ccsn_ovrf") {
                index_1("10");
                index_2("1");
                index_3("8.381359, 10.49261, 12.58327, 15.35329, 20.20066");
                values("0.648, 0.504, 0.36, 0.216, 0.072");
            }
            vector ("ccsn_ovrf") {
                index_1("100");
                index_2("1");
                index_3("53.3216, 63.67474, 70.03304, 75.49742, 82.28118");
                values("0.648, 0.504, 0.36, 0.216, 0.072");
            }
        }
        output_voltage_rise () {
            vector ("ccsn_ovrf") {
                index_1("10");
                index_2("1");
                index_3("8.480846, 10.85449, 13.2317, 16.37236, 21.8788");
                values("0.072, 0.216, 0.36, 0.504, 0.648");
            }
            vector ("ccsn_ovrf") {
                index_1("100");
                index_2("1");
                index_3("51.66086, 62.43544, 69.18598, 75.24411, 82.92101");
                values("0.072, 0.216, 0.36, 0.504, 0.648");
            }
        }
        propagated_noise_high () {
            vector ("ccsn_pnlh") {
                index_1("0.403582");
                index_2("130.649");
            }
        }
    }
}

```

```

        index_3("1");
        index_4("62.01288, 66.96726, 74.30009, 83.03548, 89.72094");
        values("0.63998, 0.591968, 0.55996, 0.591968, 0.63998");
    }
    vector ("ccsn_pnlh") {
        index_1("0.454525");
        index_2("83.1074");
        index_3("1");
        index_4("40.12712, 43.93527, 51.34704, 58.10718, 63.53206");
        values("0.602783, 0.532453, 0.485567, 0.532453, 0.602783");
    }
    vector ("ccsn_pnlh") {
        index_1("0.49882");
        index_2("59.5459");
        index_3("1");
        index_4("29.03631, 32.21923, 37.77172, 44.74476, 49.46602");
        values("0.579732, 0.495571, 0.439464, 0.495571, 0.579732");
    }
}

propagated_noise_low () {
    vector ("ccsn_pnlh") {
        index_1("0.380518");
        index_2("138.726");
        index_3("1");
        index_4("65.15771, 70.59091, 78.7748, 86.26177, 92.90636");
        values("0.0684082, 0.109453, 0.136816, 0.109453, 0.0684082");
    }
    vector ("ccsn_pnlh") {
        index_1("0.435002");
        index_2("87.9871");
        index_3("1");
        index_4("42.3349, 46.44535, 52.57853, 60.572, 65.62237");
        values("0.111277, 0.178044, 0.222555, 0.178044, 0.111277");
    }
}

ccsn_last_stage () {
    is_inverting : true;
    is_needed : true;
    miller_cap_fall : 2.168550;
    miller_cap_rise : 2.136850;
    stage_type : "both";
    dc_current ("ccsn_dc") {
        index_1("-0.72, -0.36, ...");
        index_2("-0.72, -0.36, ...");
        values("4.60473, 4.32882, ...", \
            ...
    }
    output_voltage_fall () {
        vector ("ccsn_ovrf") {

```

```

        index_1("10");
        index_2("67.0028");
        index_3("12.45957, 19.04762, 26.11126, 34.60912, 49.5931");
        values("0.648, 0.504, 0.36, 0.216, 0.072");
    }
    vector ("ccsn_ovrf") {
        index_1("10");
        index_2("201.009");
        index_3("18.57778, 37.63794, 58.13371, 82.73636, 126.6402");
        values("0.648, 0.504, 0.36, 0.216, 0.072");
    }
}

output_voltage_rise () {
    vector ("ccsn_ovrf") {
        index_1("10");
        index_2("63.7337");
        index_3("12.17514, 18.81323, 26.0398, 34.82717, 50.36987");
        values("0.072, 0.216, 0.36, 0.504, 0.648");
    }
    vector ("ccsn_ovrf") {
        index_1("10");
        index_2("191.201");
        index_3("18.30613, 37.63543, 58.53142, 84.0919, 129.7176");
        values("0.072, 0.216, 0.36, 0.504, 0.648");
    }
}

propagated_noise_high () {
    vector ("ccsn_pnlh") {
        index_1("0.404048");
        index_2("463.065");
        index_3("67.0028");
        index_4("218.894, 236.9766, 260.5929, 290.6958, 312.9011");
        values("0.643645, 0.597832, 0.567291, 0.597832, 0.643645");
    }
}

propagated_noise_low () {
    vector ("ccsn_pnlh") {
        index_1("0.373692");
        index_2("419.434");
        index_3("63.7337");
        index_4("196.2366, 213.7909, 239.5477, 262.5481, 282.7898");
        values("0.0680386, 0.108862, 0.136077, 0.108862, 0.0680386");
    }
}

}

cell_fall ("del_1_7_7") {
    index_1("2.07809, 4.71982, 10.0033, 20.5701, 41.7039, 83.9715,
168.507");
    index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,
435.754");

```

```

values("10.31793, 13.53211, 17.87725, 25.10223, 38.61968, 65.11656,
117.6417", \
      "11.42327, 14.63298, 18.97634, 26.21547, 39.7398, 66.20091,
118.6266", \
      "13.40206, 16.60506, 20.96891, 28.21968, 41.74111, 68.2456,
120.6601", \
      "16.59524, 19.85265, 24.25018, 31.53759, 45.06806, 71.58053,
123.9771", \
      "20.71864, 24.16606, 28.76576, 36.17769, 49.75945, 76.23337,
128.504", \
      "25.98608, 29.72523, 34.65855, 42.30843, 55.9951, 82.4756,
134.7845", \
      "32.61315, 36.70165, 42.21556, 50.42557, 64.48783, 91.15329,
143.5217");
}
cell_rise ("del_1_7_7") {
  index_1("2.07809, 4.71982, 10.0033, 20.5701, 41.7039, 83.9715,
168.507");
  index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,
435.754");
  values("8.989648, 12.2156, 16.6707, 24.24057, 38.60077, 66.87732,
122.8296", \
        "10.18237, 13.40198, 17.8642, 25.45202, 39.80454, 68.08088,
124.0087", \
        "12.29953, 15.51109, 19.97965, 27.58318, 41.95005, 70.34979,
126.1208", \
        "15.6698, 18.92882, 23.4234, 31.03155, 45.42379, 73.73031,
129.5502", \
        "20.21516, 23.67984, 28.3453, 36.04263, 50.40626, 78.6349,
134.5903", \
        "26.38146, 30.15188, 35.18079, 43.06213, 57.50412, 85.7551,
141.5894", \
        "34.81161, 38.96065, 44.58105, 53.02221, 67.75661, 96.20506,
152.054");
}
fall_transition ("del_1_7_7") {
  index_1("2.07809, 4.71982, 10.0033, 20.5701, 41.7039, 83.9715,
168.507");
  index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,
435.754");
  values("2.896176, 5.479719, 10.26283, 19.80126, 39.13504, 77.92579,
154.9185", \
        "2.94086, 5.524346, 10.29802, 19.80352, 39.14957, 77.87579,
155.0133", \
        "3.048865, 5.623842, 10.38063, 19.84498, 39.17374, 77.85394,
154.9042", \
        "3.592266, 6.026256, 10.61322, 19.96712, 39.25822, 77.9572,
154.8653", \
        "4.636709, 6.965882, 11.37377, 20.47392, 39.40539, 77.91851,
154.9644", \

```

```

        "6.248012, 8.533953, 12.72052, 21.42653, 39.96918, 78.3,
154.9389", \
        "8.849163, 11.09008, 15.2001, 23.45765, 41.43917, 79.16481,
155.356");
    }
    output_current_fall () {
        vector ("CCS_DRV_TEMPLATE_0") {
            index_1("83.9715");
            index_2("21.3368");
            index_3("82.38641, 87.10454, 93.93846, 97.43097, 98.70161,
102.0065, 109.5328, 114.6283, 123.0553");
            reference_time : 64.114760;
            values("-0.07928403, -0.2573575, -0.6793844, -0.8055419,
-0.8224563, -0.7516676, -0.2547098, -0.08997792, -0.01441248");
        }
    }
    output_current_rise () {
        vector ("CCS_DRV_TEMPLATE_0") {
            index_1("83.9715");
            index_2("21.3368");
            index_3("81.03659, 87.25863, 92.59878, 94.00351, 97.54884,
99.96204, 103.1054, 106.1024, 111.1576, 115.2361, 120.5518");
            reference_time : 64.114760;
            values("0.04302008, 0.2253208, 0.559771, 0.6402727, 0.7842608,
0.8035765, 0.7049112, 0.5115926, 0.2242612, 0.1040936, 0.03590279");
        }
    }
    receiver_capacitance1_fall ("CCS_RCV_TEMPLATE_0") {
        index_1("1.73331, 3.937721, 8.341532, 17.13284, 34.80908, 69.82277,
140.6975");
        index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,
435.754");
        values("1.966648, 1.968899, 1.969994, 1.970454, 1.970665, 1.970767,
1.970816", \
        "2.237084, 2.236373, 2.236784, 2.237127, 2.237322, 2.237425,
2.237478", \
        "2.377311, 2.37652, 2.376009, 2.375868, 2.375848, 2.375854,
2.375862", \
        "2.473952, 2.473492, 2.473066, 2.472715, 2.472517, 2.472419,
2.472372", \
        "2.541434, 2.543573, 2.544036, 2.543579, 2.543068, 2.542744,
2.542568", \
        "2.61439, 2.614392, 2.603713, 2.603756, 2.603814, 2.603809,
2.603788", \
        "2.681592, 2.680928, 2.679925, 2.679948, 2.680193, 2.680241,
2.680226");
    }
    receiver_capacitance1_rise ("CCS_RCV_TEMPLATE_0") {
        index_1("1.738054, 3.934024, 8.347462, 17.14354, 34.74763, 69.8089,
140.8159");
    }

```

```

index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,
435.754");
values("1.910911, 1.912998, 1.914051, 1.914503, 1.914712, 1.914813,
1.914863", \
"2.21115, 2.210694, 2.211112, 2.211466, 2.21167, 2.211778,
2.211834", \
"2.367214, 2.366509, 2.366057, 2.36595, 2.365955, 2.365975,
2.36599", \
"2.45821, 2.457763, 2.457388, 2.457074, 2.456899, 2.456815,
2.456776", \
"2.534446, 2.535101, 2.5341, 2.533768, 2.533551, 2.533417,
2.533343", \
"2.595129, 2.598873, 2.59908, 2.598835, 2.598619, 2.598917,
2.598826", \
"2.658113, 2.655571, 2.657579, 2.658597, 2.658622, 2.658532,
2.658494");
}
receiver_capacitance2_fall ("CCS_RCV_TEMPLATE_0") {
index_1("2.417703, 5.491146, 11.63803, 23.9318, 48.51935, 97.69444,
196.0446");
index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,
435.754");
values("2.702323, 2.694324, 2.693661, 2.693847, 2.694025, 2.69413,
2.694187", \
"2.758156, 2.752573, 2.748104, 2.746091, 2.745214, 2.744813,
2.744622", \
"2.973076, 2.969739, 2.965155, 2.961499, 2.959328, 2.958167,
2.957568", \
"3.389129, 3.400154, 3.403202, 3.402147, 3.400544, 3.399361,
3.398658", \
"3.73429, 3.93418, 4.062613, 4.103532, 4.129948, 4.142858,
4.149464", \
"4.363048, 4.35825, 4.30174, 4.30096, 4.351635, 4.374603,
4.390533", \
"4.31033, 4.307391, 4.3053, 4.302796, 4.280671, 4.276628,
4.290508");
}
receiver_capacitance2_rise ("CCS_RCV_TEMPLATE_0") {
index_1("2.417703, 5.491146, 11.63803, 23.9318, 48.51935, 97.69444,
196.0446");
index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,
435.754");
values("2.809006, 2.80162, 2.800809, 2.800939, 2.801096, 2.801194,
2.801247", \
"2.878897, 2.872336, 2.867637, 2.865491, 2.864556, 2.86413,
2.863929", \
"3.055137, 3.05181, 3.047371, 3.043715, 3.041503, 3.040313,
3.039698", \
"3.468901, 3.50281, 3.508573, 3.500122, 3.49942, 3.498692,
3.498216", \

```

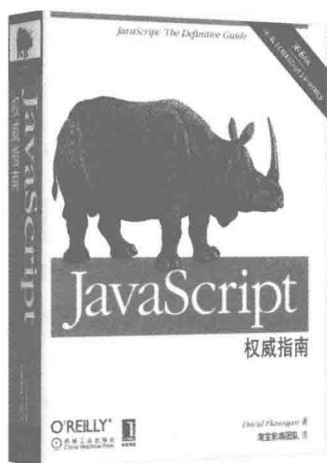
312 集成电路静态时序分析与建模

```
        "3.848895, 4.036313, 4.147219, 4.197703, 4.242004, 4.248958,  
4.255329", \  
        "4.418575, 4.415002, 4.362378, 4.361294, 4.404152, 4.428536,  
4.439927", \  
        "4.368137, 4.369721, 4.364221, 4.358307, 4.330571, 4.342502,  
4.352572");  
    }  
    rise_transition ("del_1_7") {  
        index_1("2.07809, 4.71982, 10.0033, 20.5701, 41.7039, 83.9715,  
168.507");  
        index_2("0.001, 7.15153, 21.3368, 49.4777, 105.304, 216.052,  
435.754");  
        values("2.662085, 5.441622, 10.59789, 20.99208, 41.98968, 84.08149,  
2.711295, 5.47692, 10.63972, 20.98717, 42.03585, 84.17687,  
2.814971, 5.584172, 10.69458, 21.02007, 42.09063, 84.13484,  
3.362606, 5.985388, 10.94285, 21.12609, 42.1321, 84.15716,  
4.393543, 6.903294, 11.64204, 21.55804, 42.30236, 84.25525,  
6.070472, 8.535403, 13.02515, 22.48419, 42.72635, 84.51585,  
8.771537, 11.20973, 15.6252, 24.48684, 44.12034, 85.24997,  
168.281");  
    }  
}  
}  
}  
default_operating_conditions : "BALANCED";  
default_wire_load : "DEFAULT";  
}
```

参考文献

- [1] 魏少军. IC 设计业: 未来仍须着重提升内功[D]. 北京: 清华大学出版社, 2009.
- [2] 李国杰. 发展 IC 产业应重视的几个问题[D]. 北京: 科学出版社, 2008.
- [3] 徐靖. X 微处理器的半定制 / 全定制混合设计研究[D]. 长沙: 国防科技大学出版社, 2009.
- [4] 冯超超. 半定制 / 全定制混合设计流程中验证方法研究[D]. 长沙: 国防科技大学出版社, 2007.
- [5] 王延宁. 面向全定制与半定制混合设计方法的噪声分析与设计[D]. 长沙: 国防科技大学出版社, 2007.
- [6] Jos Budi Sulisty. On the Characterization of Library Cells[D]. Virginia: the Virginia Polytechnic Institute and State University Press, 2000.
- [7] 张年翔. Cell-Based IC Physical Design and Verification-SOC Encounter[P]. CIC, 2006.
- [8] 陈麒旭. Static Timing Analysis[D]. CIC, 2008.
- [9] J B hasker, Rakesh Chadha. Static Timing Analysis for Nanometer Designs[M]. Berlin: Springer, 2009.
- [10] Shuo Zhou. Static Timing Analysis in VLSI Design[D]. California: university of California Press, 2006.
- [11] Sravanth Valluri. Static Timing Analysis[R]. Chandigarh: Semiconductor Complex Limited, 2006.
- [12] Timing Analysis Basics[R]. Cadence, 2012.
- [13] EncounterTM Library Development Guide[R]. Cadence, 2016.
- [14] Encounter Timing System User Guide[R]. Cadence, 2016.
- [15] Encounter Library Characterizer User Guide[R]. Cadence, 2016.
- [16] PrimeTime User Guide[R]. Synopsys, 2016.
- [17] Synopsys. PrimeTime SI User Guide[R]. Synopsys, 2016.
- [18] Synopsys. PrimeTime SI Workshop Student Guide[R] USA: Synopsys, 2016.
- [19] Synopsys. Library CompilerTM User Guide[R]. Synopsys, 2016.
- [20] Synopsys. SiliconSmart User Guide[R]. Synopsys, 2016.
- [21] Synopsys. PrimeTime Introduction to Static Timing Analysis Workshop[R]. Synopsys, 2016.
- [22] Brent B Welch. Tcl/Tk 编程权威指南(第 3 版)[M]. 北京: 中国电力出版社, 2002.
- [23] Ray Johnson. Tcl Style Guide[J]. Sun Microsystems, 1999.
- [24] Brent B Welch, Ken Jones, Jeffrey Hobbs. Practical Programming in Tcl and Tk, Fourth Edition[M]. New Jersey: Prentice Hall Publication, 2003.
- [25] 洛阳城主. Tcl 语言教程[C]. 洛阳: 河南科技大学出版社, 2011.
- [26] 华为. TCL 培训教程[P]. 深圳: 华为技术有限公司, 2005.

推荐阅读



JavaScript权威指南（原书第6版）

从1996年以来，本书已经成为 JavaScript 程序员的《圣经》。

第6版涵盖HTML5和ECMAScript 5。很多章节完全重写，以便跟上当今的最佳Web开发实践。该版本的新增章节描述了jQuery和服务端JavaScript。

对于那些希望学习Web编程语言的有经验的程序员和希望精通JavaScript的当前JavaScript程序员，本书最适合不过了。

深入理解PHP：高级技巧、面向对象与核心技术（原书第3版）

国际知名Web开发专家和技术畅销书作家最新力作，PHP领域经典著作。

从编程技巧、面向对象和扩展三个角度系统讲解和总结了成为中高级PHP程序员应该具备的技术和技能，包含大量实用案例，极具实践指导意义。

如果你已经具备一定的PHP编程基础，想使开发效率更高，想把应用做得更好，那么这本书应该是你需要阅读的。本书旨在为想修炼成为高级PHP程序员的中初级PHP程序员提供实用的方法和建议。

推荐阅读



Effective JavaScript: 编写高质量JavaScript代码的68个有效方法

作者: David Herman ISBN: 978-7-111-44623-1 定价: 49.00元



JavaScript编程精解

作者: Marijn Haverbeke ISBN: 978-7-111-39665-9 定价: 49.00元



HTML5 WebSocket权威指南

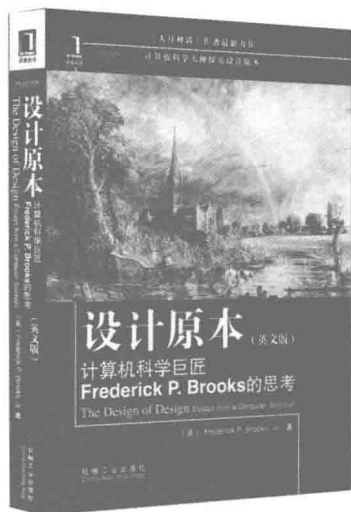
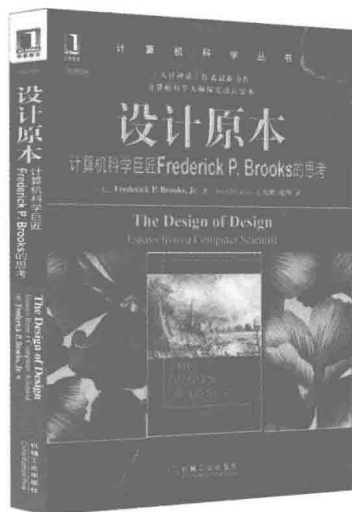
作者: Vanessa Wang 等 ISBN: 978-7-111-45641-4 定价: 49.00元



构建实时Web应用: 基于HTML5 WebSocket、PHP和jQuery

作者: Jason Lengstorf 等 ISBN: 978-7-111-43983-7 定价: 69.00元

推荐阅读



设计原本——计算机科学巨匠Frederick P. Brooks的思考

作者: Frederick P. Brooks 译者: InfoQ中文站 等 ISBN: 978-7-111-32557-4 定价: 55.00元

设计原本——计算机科学巨匠Frederick P. Brooks的思考 (英文版)

作者: Frederick P. Brooks ISBN: 978-7-111-32503-1 定价: 69.00元

《人月神话》作者最新力作 计算机科学大师探究设计原本

深入理解计算机系统 (原书第2版)

作者: Randal E. Bryant等 ISBN: 978-7-111-32133-0 定价: 99.00元

深入理解计算机系统 (英文版·第2版)

作者: Randal E. Bryant等 ISBN: 978-7-111-32631-1 定价: 128.00元

Linux内核设计与实现 (原书第3版)

作者: Robert Love ISBN: 978-7-111-33829-1 定价: 69.00元

Linux内核设计与实现 (英文版·第3版)

作者: Robert Love ISBN: 978-7-111-32792-9 定价: 69.00元

集成电路静态时序设计入门必读的一本书

集成电路静态时序设计与它的工程应用密切相关，大多数高校不具备静态时序分析所需要的工程环境，因此注定集成电路静态时序设计人才难求，薪资也普遍位列IC行业前茅。很多集成电路设计专业毕业的研究生在选择工作岗位的时候，由于对静态时序设计没有工程应用方面的实际经验，因此不能得到高薪的时序设计岗位。本书不但能够让没有接触过静态时序设计的工程师和研究生对静态时序设计的技能、知识有一个系统的了解，还能使他们结合静态时序设计所需要的工具进行实践，快速掌握静态时序设计工作的基本技能。

本书特点

- 本书在广度上全面覆盖集成电路静态时序设计的两个重要设计大方向：静态时序分析和静态时序建模。同时在深度上覆盖两大重要设计方向相互关联的技术点以贯穿整个静态时序设计流程，使读者在广度和技术点衔接两个方面深入理解整个静态时序设计技术和流程细节。
- 本书基于完整工程设计的角度，针对整个集成电路静态时序设计过程，通过结合业内主流EDA设计工具和实例操作的形式进行讲解，最终以理论联系实际的方法真正提高读者学以致用用的工程技术设计能力。
- 本书的实验技术资料可从www.hzbook.com下载。



上架指导：集成电路设计

ISBN 978-7-111-53777-9



9 787111 537779 >

定价：79.00元

投稿热线：(010) 88379604

客服热线：(010) 88378991 88361066

购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com

网上购书：www.china-pub.com

数字阅读：www.hzmedia.com.cn